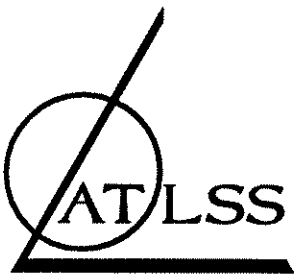
8-1-1994

Concepts and Notation for Integrated Structural Design: Product and Process Models

Namhee K. Hong

Richard Sause



ADVANCED TECHNOLOGY FOR
LARGE
STRUCTURAL SYSTEMS

Lehigh University

CONCEPTS AND NOTATION FOR INTEGRATED STRUCTURAL DESIGN: PRODUCT AND PROCESS MODELS

by

Namhee K. Hong
ATLSS Research Assistant
Department of Civil Engineering
Lehigh University

Richard Sause
Assistant Professor of Civil Engineering
Lehigh University

ATLSS Report No. 94-13

August 1994

ATLSS Engineering Research Center
Lehigh University
117 ATLSS Dr., Imbt Laboratories
Bethlehem, PA 18015-4729
(610) 758-3525

An NSF Sponsored Engineering Research Center

Acknowledgement

This research has been supported by the Engineering Research Center for Advanced Technology for Large Structural Systems (ATLSS) at Lehigh University under NSF Grant ECD-8943455. Opinions, findings, and conclusions are those of the authors and do not necessarily reflect the views of NSF.

Table of Contents

List of Figures

Abstract

Integrated computer-aided design systems manage, communicate, and process the information created by and used in a variety of planning and design activities. Formal models of both the design product and the design process are important conceptual steps in the development of an integrated system. The report presents research toward integrated product and process models for structural design in which each entity in the product model describes information used by activities in the process model, and each activity in the process model is described as operations on certain entities in the product model. The report focuses on developing formal and consistent concepts and notation for use in creating integrated product and process models for structural design based on the concepts of semantic data models and related abstraction mechanisms. The formal concepts and notation enable the design information and the design activities to be tied together. Previous research on formal models for structural design is reviewed. Information modeling concepts useful for developing product and process models are studied. Then, elements and notation needed to create integrated product and process models are developed. An approach to generate product and process models for structural design using these elements and notation is developed. Finally, the product and process modeling approach is applied to the preliminary design of beams. The report concludes with a summary and recommendations for future research.

1. Introduction

In order to advance beyond the current state of computer-aided design systems for structural engineering, there is a need for formal models to describe and organize design information and design activities (Martini et al. 1991; Neville et al. 1989; Sause and Powell 1990). The research presented in this report focuses on the development of these models, termed integrated product models and process models, for structural design. In this context, the term models refers to things that are used to describe design information and design activities. A product model provides a formal framework for describing and organizing design information. Similarly, a process model provides a formal framework for describing and organizing design activities. Integrated product and process models are models in which the product model describes design information used by design activities in the process model, and the process model describes design activities as operations on certain information in the product model.

The remainder of this chapter is organized as follows. Section 1.1 discusses recent research on computer-aided design systems. Section 1.2 identifies problems that have not been adequately addressed by previous research. Section 1.3 outlines the objectives of the research presented in this report. Finally, Section 1.4 discusses the scope of this research and the research approach.

1.1 Recent Research

Advanced computer science technologies, such as artificial intelligence, expert systems, data modeling techniques (e.g., semantic data modeling, object-oriented data modeling), symbolic programming languages (e.g., Prolog, Lisp, etc.), object-oriented programming languages (e.g.,C++, Smalltalk, etc.), and so on, have had a significant impact on the development of computer-aided design systems in engineering. Research on computer-aided design systems has been in two main areas.

The first research area seeks to improve some drawbacks of conventional computer-aided design systems, for example, by increasing the level of domain-specific knowledge embedded in the system (i.e., by using knowledge-based systems), by providing user-friendly interfaces, by using

3-D graphics for design visualization, and so on. Most research in this area has focused on the application of advanced programming technologies from computer science, rather than on the application of formal information modeling concepts and tools. As a result, most software for computer-aided design has been developed in areas such as analysis, member design, and drafting, where well-defined information models already exist. Areas such as conceptual design, where well-defined information models are not available, have not been adequately addressed.

The second area of research seeks to develop computer-aided design systems that can support different design activities, such as structural analysis, component design, drafting, cost evaluation, and so on, within one system. The goals of this type of computer-aided design systems are to keep consistent information flow from one design activity to another, to eliminate trivial intermediate design tasks, to control data redundancy, and to obtain data integrity. Most research in this area has focused on the integration of design activities by way of data translation or data transformation, rather than on formal information modeling. Models that integrate the full range of design information for constructed facilities have not yet been developed.

In order to advance the current state of computer-aided design, four broad steps are involved: (1) identify concepts used in design, (2) develop formal information models for the concepts used in design, (3) develop implementation models for the concepts used in design, and (4) develop computer software. Figure 1.1 shows these four steps on the left side and the results of each step on the right side. In order to develop integrated computer-aided design systems, each step should receive sufficient attention. To date, sufficient attention has not been paid to the first two steps.

The primary purpose of this report is to develop and present formal modeling concepts and notation needed during the initial steps (i.e., steps (1) and (2) in Fig. 1.1) toward integrated computer-aided design systems for structural engineering.

1.2 Problem Statement

Previous research on computer-aided design systems has not adequately addressed several issues. It is believed that by addressing these issues, better models for design information and design activities (i.e., design product and process models) for structural engineering can be developed.

These issues are as follows:

1. A consistent terminology for product and process models is not available.

2. Modeling concepts and notation that apply to both design information and design activities are not available. Existing methodologies for developing product models (e.g., the network model, semantic model, relational model, etc.) are suitable mainly for describing design information because these methodologies do not consider how the data is used and generated by various design activities. Existing methodologies for developing process models (e.g., the data flow diagram, IDEF0, etc.) are suitable mainly for describing activities because these methodologies do not consider how the data used by the activities should be organized in the model.

3. Models that tie design information and design activities together have not been developed. Existing models focus on the representation of either design information or design activities using different modeling concepts respectively.

4. Models that organize design activities in a formal way are not available. Existing models for design activities focus more on the representation of the overall information flow of the design process than on the organization of design activities themselves.

5. Suitable modeling concepts and tools for early design stages (e.g., conceptual and preliminary design stages) have not been developed. Generally, the purpose of conceptual design is to identify a set of feasible solutions (i.e., design alternatives) to the design problem and the purpose of preliminary design is to establish which of the design alternatives is the best solution. That is, models for early design stages must be capable of representing the information and activities involved in the development and comparison of several design alternatives. However, models that manage structural design alternatives in a consistent way have not been developed.

1.3 Objectives

The overall objective of the research is to develop formal and consistent modeling concepts and notation for describing and organizing design information and design activities (i.e., for developing design product and process models). More specifically, the modeling concepts and notation are entity-based, which allows us to develop product and process models using semantic

data modeling concepts. These modeling concepts and tools will support the development of integrated computer-aided design systems for structural engineering. Specific objectives of the research are to:

1. Define the consistent terminology (i.e., elements of product and process models) needed to develop modeling concepts and notation for structural design.

2. Develop formal and consistent concepts and notation for describing and organizing design information and design activities. These modeling tools will be capable of representing both design information and design activities, and will tie the design information to the design activities.

3. Demonstrate the utility of the modeling concepts and notation by applying them to several examples.

1.4 Scope and Approach

There are many kinds of research needed to develop integrated computer-aided design systems. The research described in this report is limited to the development of design product and process models as follows. The design product model consists of entity categories that are used to describe a structural system during its design; and the design process model consists of entity categories that are used to describe the design activities involved in the design of a structural system. The design product model describes design information that directly supports design activities in the design process model. In addition these design product and process models focus on the representation of design alternatives for structural systems (e.g., beams and frames) during the preliminary design stage. However, the overall control of design process and the description of knowledge used in the design process to make decisions is not the main focus of these models. The research in this report focuses on product and process models that represent design alternatives for structural components (e.g., beams and columns) during the preliminary design stage.

In order to accomplish the research objectives within the scope of this study, the following approach is adopted:

1. Review information modeling concepts developed in computer science to define a consistent terminology for product and process models for structural design. In particular, semantic data

models, including the concepts of entities, relationships, attributes, and the appropriate abstraction mechanisms, are reviewed because these concepts are sufficiently general to be used in describing and organizing various kinds of information.

2. Study the structural design process, focusing on design alternatives, to identify the requirements for product and process models.

3. Develop formal and consistent modeling concepts and notation to represent both design information and design activities.

4. Develop specific product and process models to demonstrate how to use the concepts and tools.

The remainder of this report is organized as follows. Chapter 2 reviews previous design model research related to the research presented in this report. Chapter 3 reviews information modeling concepts to develop specific modeling concepts and notation for use in structural design product and process models. Chapter 4 defines the elements of product and process models for structural design. Graphical conventions for these models are also discussed in Chapter 4. Chapter 5 presents a general approach for developing product and process models for structural design. Specific product and process models for the preliminary design of beams are then presented in Chapter 6. Finally, Chapter 7 summarizes the report and presents some concluding remarks.

2. Previous Design Model Research

The purpose of this chapter is to discuss previous research on models for design information and activities, and to introduce several fundamental ideas about models for structural design. The chapter is organized as follows. First, Section 2.1 introduces two types of formal models for structural design, namely, product models and process models, and review previous research on product and process models respectively. Then, Section 2.2 defines several terms related to these models. Finally, relationships between the two formal models are discussed in Section 2.3.

2.1 Formal Models for Structural Design

Formal models for structural design have been studied recently by several investigators (Howard et al 1991; Luth et al 1991; Martini et al. 1991; Sause and Powell 1990, 1991). Sause et al. (1992) identified two types of formal models that are important for structural design: (1) *product models* that describe the structural systems being designed, and (2) *process models* that describe the activities that comprise the design process. As an introductory example, Figure 2.1 shows a product model and a process model for plate girder design. The figure illustrates how information about plate girder design is mapped to a product model and a process model respectively. Plate girder design properties, such as web thickness, web depth, flange thickness, flange width, steel grade, and so on, are represented in a product model; and plate girder design tasks, such as calculate the moment diagrams, calculate the maximum shear force, propose the web depth, propose the flange thickness, review the web for shear, and so on, are represented in a process model.

2.1.1 Previous Research on Product Models

Product models for engineering design have been developed for several purposes such as the development of database systems, the exchange of data between design and construction activities, and so on. Research on product models for engineering database systems has relied on data modeling concepts developed in the domain of computer science. Examples related to product

models for engineering database systems include the component-connection abstraction model (Powell and Bhateja 1988), the engineering data model (Eastman et al. 1991), and the building design data model (Law et al. 1990). Data exchange is currently a national and international concern. The STandard for the Exchange of Product Model Data (STEP) is a recently developed data exchange standard for product models. Two formal information modeling techniques, NIAM and IDEF1x are used to develop a STEP model. The basic constructs of IDEF1x language are the entity, the attribute, and the relationship, and those of NIAM are the object and the relationship. PDES, Product Data Exchange using STEP, is an activity in the United States that supports the development and implementation of STEP (IPO 1991). Under the PDES, GARM, general AEC reference model, was developed to provide a general data reference model for all AEC application areas (Gielingh 1988). GARM is based on one generic entity, which is the product definition unit (PDU). GARM uses four fundamental methods for describing the characteristics of PDUs: specialization, decomposition, characterization, and life cycle. Example product models for data exchange include the RATAS model (Bjork 1989), and the primitive-composite (P-C) data model (Howard et al. 1992). The RATAS model was developed as a standard CAD data model for the Finnish national construction industry. This model describes a building using objects (i.e., entities) and organizes these objects in a network of relations between objects. The P-C data model uses object-oriented concepts to represent primitive things in a domain and to combine these primitive things to make the composite things that engineers typically consider.

In general the development of the product models for database systems or data exchange has focused on the representation of information about a completed design rather than on the representation of information during the design process (which is the one of main concerns of this report). However, the P-C approach provides a starting point for organizing design information and activities toward the development of design product and process models. The following briefly discusses the P-C approach.

Howard et al. (1992) proposed the "Primitive-Composite (P-C)" approach for modeling design information. The main objective of the work that led to the P-C approach was to enable efficient data exchange between participants in the domain of AEC. The P-C approach uses two types of entities to represent design information: (1) "primitive entities" and (2) "composite entities". A primitive entity represents an atomic concept of a real-world object, such as shape, material, behavior, and so on. The primitive entities are individual things that can be used in the

composition of many different composite entities. The composite entities are made by aggregating the primitive entities to represent the things that engineers are typically concerned with (e.g., walls, columns, beams, and so on). Figure 2.2 illustrates schematically how composite entities are made by aggregating primitive entities, each of which is defined independently. Different participants in the domain of architecture-engineering-construction will use different composite entities to represent the same real-world objects. For example, an architect and an engineer would use different composite entities to represent the same wall. The primitive entities that are common to the related composite entities are a medium for data exchange between different participants.

2.1.2 Previous Research on Process Models

Most of the previous research on process models for engineering design has focused on the flow of information in the design process using functional modeling concepts developed in the domain of computer science. For example, the integrated building process model (IBPM) has been developed to integrate the management, planning, design, construction, and operation of constructed facilities for the construction industry using a modified IDEF0 model (Sanvido et al. 1989). The modified IDEF0 model used in the IBPM represents functions and data flow. There are four different types of data flow: inputs, outputs, mechanisms, and controls. Another example is a, "partitioned engineering data flow" (PANDA) model, which was developed to provide a reference model for functional analysis in facility engineering using an extension of the data flow model (Phan and Howard 1994). The fundamental concepts of the extended data flow model used in the PANDA are similar to those of the modified IDEF0 discussed above. PANDA has three major partitions (i.e., (1) participants, (2) process, and (3) data, material and products). The extended data flow diagram of PANDA is structured according to these three major partitions.

Other research on process models for structural design has produced the so-called "Multilevel Selection Development" (MSD) model (Sause and Powell 1990, 1991). The MSD model is an organizational model for the process of structural design and represents the design process as a hierarchy of *selection* and *development* activities, in which: (1) selection involves identifying, ranking, eliminating, and selecting from a number of competing alternatives; and (2) development involves design and evaluation of the design alternatives. The basic ideas of the MSD model are illustrated in Figure 2.3. The MSD model is a starting point for the current research on design

process models. It focuses more on organization of design activities than on information flow between activities in the design process.

2.1.3 Previous Research on Integrated Product and Process Models

Previous work on product and process models has focused on the representation of either design information (in a product model) or design activities (in a process model). Recently, Sause and Madden (1993) proposed integration of these models into a single model for structural design. In these integrated product and process models, each entity in the product model describes information created and used by activities in the process model, and each activity in the process model is described as operations on entities in the product model. Sause and Madden outlined a few integrated product model entities and process model activities, but did not represent the entities and activities, or the relationships between them in a formal way.

2.2 Generalization Models and Instance Models

Two types of formal models (i.e., design product and process models) are further defined based on the concepts of a *generalization model* and an *instance model*. The term "model may refer to either a generalization model or an instance model. A model which provides a framework or generalized description that is used to describe the things in a particular domain of interest is called a *generalization model* (Sause et al. 1992).

A generalization model consists of categories and relationships that can be used to describe certain types of products and processes in a domain; a generalization model should be defined for the domain of interest. For example, a generalization model for structural design provides categories of concepts for the design of structural systems as well as the relationships among these categories of concepts. A model of a particular product or process is called an *instance model* (Sause et al. 1992). For example, an instance model for structural design describes the design of a specific structural system. The current report focuses first on the development of structural design generalization models including the categories and relationships that compose these models. Then, structural design instance models are considered.

This research is concerned with two types of generalization models for structural design: (1) *design product generalization models* which consist of categories that are used to describe a structural system during its design; and (2) *design process generalization models* which consist of categories that are used to describe the activities involved in the design of a structural system. Similarly, there are two types of structural design instance models: (1) a *design product instance model*, which describes a particular structural system during its design (e.g., the structural system used in a particular project); and (2) a *design process instance model*, which describes the design process for a particular structural system (e.g., the design process for one particular structure). A design product instance model uses instances from the categories of a design product generalization model to describe a particular design product. A design process instance model uses instances from the categories of a design process generalization model to describe a particular design process.

Figure 2.4 shows an example of a design product generalization model for beam design. This generalization model is composed of several categories, such as wide flange beam, plate girder, open web bar joist, and so on. If four beam design alternatives are considered in the design of a particular beam, an instance model that has four instances from the categories of the generalization model in Figure 2.4 is created as shown in Figure 2.5.

Through the remainder of this report, design product generalization models and design process generalization models are simply called *product models* and *process models*, respectively; design product instance models and design process instance models are simply called *product instance models* and *process instance models*, respectively.

2.3 Relationships Between Product and Process Models

As discussed previously, product models and process models provide categories for describing design information and activities. A product model provides categories for describing a structural system during its design, a process model provides categories for describing the design process for the structure. Since both models are needed to design structural systems, the categories of the product model should be related to the categories of the process model. That is, information described by categories of the product model is used and generated by the activities described by

the categories of the process model. For example, the information described by the "plate girder entity" category of the product model in Figure 2.1 is used and generated by the tasks by the "plate girder activity" category of the process model in Figure 2.1. The concept of integrated product and process models draws on this kind of relationship between the product and process models. The way of formalizing the integration relationship between the two different models is discussed in Chapter 5.

3. Information Modeling Concepts

This chapter presents an overview of information modeling concepts from the domain of computer science that are useful for organizing information and activities in formal product and process models for structural design. These information modeling concepts include entities, relationships, attributes, and abstraction mechanisms. The concepts come from previous research on "semantic data models" (Hammer and McLeod 1981; Hull and King 1987; Thompson 1989). A detailed discussion of semantic models is beyond the scope of this report. The chapter outlines information modeling concepts that are not limited to any specific purpose (e.g., database design, the development of programming languages, etc.). The concepts are sufficiently general to be used in describing and organizing information about any physical object.

The chapter is organized as follows. In Section 3.1, semantic models are briefly introduced. The difference between semantic models and "nonsemantic models" is discussed through several examples. Section 3.2 outlines benefits that can be achieved using the concepts of semantic models to organize structural design information. In Section 3.3, the primary components of semantic models are described. These components include entities, relationships, and attributes. In addition, the term "entity category" is defined. Finally, Section 3.4 describes the three types of abstraction mechanisms that are provided by semantic models. Additional terms, such as "class", "instance", "supercategory", "subcategory", "compound category" and "simple category" are defined in the discussion of the abstraction mechanisms.

3.1 Semantic Models

A model that can capture the semantics of data is called a semantic model. The term "semantic" refers to the meaning of data. A semantic model conveys the meaning of the data explicitly. The first semantic model was the entity-relationship (E-R) model developed by Chen (1976). In the E-R model, the entities and relationships of a model are defined using E-R diagrams. The terms "entities" and "relationships" will be discussed later on.

Semantic models were developed to overcome limitations of the record-based approach on which

traditional data models (e.g., relational models, hierarchical models, and network models) are based. The data structures used in the traditional models are relatively close to those used for physical representation of data in computers, ultimately viewing data as collections of records with printable or pointer values (Hull and King 1987). These traditional models are referred to as "nonsemantic" models in this report. In contrast, semantic models represent data in ways that are more directly correlated to physical objects in the real world. This is accomplished by providing "abstractions" for modeling data (e.g., entities and relationships). These abstractions (rather than records containing printable or pointer values) are the fundamental means for representing information. They allow users to focus more directly on abstract models for real-world objects rather than on printable or pointer values.

The graphical diagrammatic conventions provided by most semantic models are used to express the meaning of the data. For example, the E-R model provides two graphical conventions; rectangles for entities, and diamonds for relationships. Another graphical convention has been developed by Thompson (1989) for semantic data models, which is called the "semantic data model diagram". Graphical diagrammatic conventions are discussed in detail in Chapter 4.

Figures 3.1 and 3.2 illustrate the difference between a semantic model and a nonsemantic model. An E-R model for a beam is shown in Figure 3.1. In this model, the entities, relationships, and attributes are clearly indicated by: (1) rectangles for entities, such as BEAM, MATERIAL, and GEOMETRY; (2) diamonds for relationships, such as is_part_of; and (3) bars with small circles at the ends for the attributes, such as section_area, moment_of_inertia, material_density, and elastic_modulus. The model describes the information about beams using BEAM entities, MATERIAL entities, GEOMETRY entities, and is_part_of relationships, rather than using only printable data. The model clearly indicates that a beam is viewed as having a geometry and a material as its components. The information about the beam's material and geometry is then described using material and geometry entities and their respective attributes. The use of the entities and relationships allows the complexities of information to be represented effectively.

Figure 3.2 is a relational model that has the information shown in Figure 3.1 in a so-called "relational table". A relational model is a nonsemantic model. In a relational model, one entity is represented as one or more records in one or more tables (i.e., information for one entity may be distributed over several tables). This record-based representation does not identify entities and

relationships explicitly. Identifying entities and relationships is an initial step in developing a model for information about real-world objects. However, a record-based representation makes it difficult to represent these entities and relationships explicitly.

The problem can be illustrated by converting the information for a beam shown in Figure 3.1 into relational tables as in Figure 3.2. Figure 3.2c is a relational table for a beam, which can be obtained from the other relational tables in Figure 3.2 (i.e., the MATERIAL relational table in Figure 3.2a and the GEOMETRY relational table in Figure 3.2b) by a relational operation "join" (Date 1981). In the relational model, two tables may be joined over two columns that are defined over a common domain. For example, the tables in Figures 3.2a and 3.2b may be joined over the columns beam_name. The result of the join is a new and wider table (Figure 3.2c) which describes the beam more completely. However, the BEAM relational table in Figure 3.2c represents the beam with printable data, and obscures the relationships between the BEAM, MATERIAL, and GEOMETRY entities. Thus, it is difficult to see from this model that a beam is viewed as having material and geometry as its components, because the relationships are not explicitly shown by the tables in Figures 3.2.

3.2 Use of Semantic Models for Structural Design Information

Semantic models provide several advantages in representing structural design information. In this chapter, two benefits of semantic models are discussed: (1) the clear mapping between the components of a structural system and the corresponding entities in the information model that can be achieved, and (2) the structured classifications of structural design information that can be developed, usually in the form of entity hierarchies.

The first benefit is obtained by the use of the abstractions of data provided in a semantic model. These data abstractions are the concepts of "entity", "relationship", and "attribute" provided by semantic models, which allow information to be represented without concern for the details of data representation in the computer (i.e., records, and printable or pointer values). Thus, entities in the model can correspond rather directly to the components of an existing system. For example, a frame can be represented in terms of a frame entity, beam entities, and column entities (Figure 3.3). The specific information (e.g., geometric properties of members, material density, etc.) associated

associated with the beams and columns that compose the frame can be added to this basic model as needed, however, it is possible to develop the model of Figure 3.3 without concern for the details of this additional information.

The second benefit is obtained by using the abstraction mechanisms that are available in a semantic model. Abstraction mechanisms are used to emphasize the essential properties of a set of entities and neglect their differences. Three important abstraction mechanisms are classification/instantiation, generalization/specialization, and aggregation/decomposition. These abstraction mechanisms can be combined with each other to create a hierarchical structure for complex sets of information.

Classification/instantiation provides the membership relationship between a class and the instances or members of a class. Generalization/specialization enables the categorization of engineering objects from the most general level to the most specific level. Aggregation/decomposition abstraction allows the entities to be assembled from other entities. These abstraction mechanisms are discussed in more detail in Section 3.4.

3.3 Entities, Relationships, and Attributes

The primary components of semantic models are entities, relationships, and attributes. A "category" is a set of things that are naturally grouped together because of their similar properties. In this report, an "entity category" is a set of entities that have similar properties. Semantic models include various types of entity categories that are constructed by using abstraction mechanisms. In this section, entities, relationships, and attributes are discussed. The various types of entity categories and abstraction mechanisms are discussed in the following section.

The terms entity, relationship, and attribute were originally defined in the E-R model (Chen 1976). The E-R model explicitly separates information about entities from information about relationships. An *entity* is defined as a "thing" which can be distinctly identified. In general, entities are classified into different entity categories according to their properties. For example, entities for the domain of structural design could be classified into different entity categories, such as FRAME, BEAM, COLUMN, CONNECTION, and so on, as shown in Figure 3.3b.

Entities of each entity category may be further categorized into other entity categories according to

their more specific properties. That is, one entity category may contain many lower level entity categories. The reverse is also true. That is, many categories may be grouped into an upper level entity category. For example, entities of the BEAM category in Figure 3.3b may be categorized into lower level entity categories that contain different kinds of beams, such as WIDE FLANGE BEAM, PLATE GIRDER, REINFORCED CONCRETE BEAM and so on, as shown in Figure 3.3c.

The description of an entity category defines the properties common to all the entities in the entity category. For example, the properties section_area and moment_of_inertia of the entities in the BEAM category (Figure 3.1) could be common properties for a wide flange beam, a plate girder, a reinforced concrete beam, and so on. The entities in an entity category may have more specific properties in addition to the common properties. For example, a wide flange beam has a more specific property, called shape_designation, in addition to section_area and moment_of_inertia.

A *relationship* is an association among entities. A "relationship category" is a collection of relationships. For example, is_connected_to in Figure 3.4a is a relationship category for relationships between entities of the BEAM category and the COLUMN category. Relationship categories are characterized by a cardinality to show the number of relationships that can exist for each member of the entity categories. The cardinality is usually shown with the relationship category as in Figure 3.4b. The cardinality for the relationship category is_connected_to indicates that many beams are connected to many columns and the cardinality for the relationship category is_part_of between the FRAME and the BEAM categories indicates that one frame has many beams.

Entity and relationship categories are characterized by *attributes*. Attributes represent elementary properties of entities and relationships. For example, section_area and moment_of_inertia are the attributes of the entities of the BEAM category, as shown in Figure 3.1. The relationships of the relationship category is_connected_to between the BEAM category and the COLUMN category could have attributes such as number of members and connection types. Attributes are also characterized by cardinality to show the number of values that an attribute can have at one time (i.e., one attribute may have more than one value).

3.4 Abstraction Mechanisms

The abstraction mechanisms described here have been widely used to help developers of database systems to understand, classify, and model information. Abstraction can be defined as a mental process that we use to select certain characteristics of a set of entities that are relevant and to exclude other characteristics that are not relevant (Batini 1992). The abstraction mechanisms provided by semantic models are used to focus on the characteristics of a set of entities that are regarded as essential and to neglect the differences among the entities.

Three types of abstraction are used to develop models for structural design: classification, generalization, and aggregation. These three abstraction types are considered to be pairs of opposite processes (i.e., classification/instantiation, generalization/specialization, and aggregation/decomposition), which are graphically represented as one-level trees (Figures 3.5, 3.6, and 3.8). These are discussed in more detail in the following three sections.

3.4.1 Classification and Instantiation

A classification abstraction is used for defining a "class" for a set of entities. The members of a class have identical attributes. The class itself becomes a single concept that refers to the members of the class (Batini 1992). If the entities in a specific entity category are described by a single concept, the entity category is called a "class" and its members, which all have the same attributes, are called "instances". The opposite process of classification is called instantiation. Each instance of a class is created by "instantiating" the class. When instantiation is performed, the class behaves like a template to generate the instance. All instances have the same attributes, which are defined by the class. However, each instance has its own values for the attributes.

As defined in this report, a class differs from an entity category in two ways: (1) a class describes entities whose attributes (but not the attribute values) are identical while an entity category describes entities that may have additional attributes in addition to the common attributes defined by the category; and (2) entities of a class are not further categorized into lower level subclasses while those of an entity category may be categorized into lower level entity subcategories, as discussed in Section 3.3.

Figure 3.5a shows a set of entities whose attributes are all identical. Application of classification abstraction to the set of entities in Figure 3.5a results in a single level classification/instantiation tree, as in Figure 3.5b. Classification defines the root entity of the tree (i.e., a class) which describes the attributes of the members (i.e., instances) of the class. Instantiation generates the leaf entities of the tree (i.e., the instances) which are actual occurrences of the class. The arrows between the class and the instances represent the pair of relationships member_of/prototype_of depending on their directions.

Note that Figure 3.5b includes: (1) a value set for each attribute in the class definition to indicate the domain of the possible values of the attribute; and (2) attribute values for each attribute of the instances. Value sets are discussed in Chapter 4. Figure 3.5c illustrates an example of classification abstraction for wide flange beams. This was shown as the WIDE FLANGE BEAM category in Figure 3.3c. In Figure 3.5c, WIDE FLANGE BEAM is the class whose members are all wide flange beam shapes (i.e., W16*40, W14*38, etc.).

3.4.2 Generalization and Specialization

A generalization abstraction defines a subset relationship between two or more entity categories (Batini 1992). The term generalization was introduced to the area of database design by Smith and Smith (1977). Based on their definition, generalization can be described as follows: an entity category E is a generalization of a group of entity categories E1, E2, ..., En if each member of entity categories E1, E2, ..., En is also a member of an entity category E.

Generalization abstraction enables the creation of generalization hierarchies. The use of generalization hierarchies simplifies the definition of new categories, because the attributes of a supercategory are inherited by its subcategories, and a new category can often be defined as a subcategory of an existing category. The opposite process of generalization is specialization. To specialize a category (i.e., to create subcategories), other attributes that are pertinent to each subcategory under consideration are added.

Figure 3.6a shows a set of entity categories that have the common attributes. Application of generalization abstraction to the set of entity categories in Figure 3.6a results in a single level generalization/specialization tree, as in Figure 3.6b. Generalization defines the root entity category

of the tree (i.e., the supercategory) which summarizes the common attributes of the entity categories in the tree. Specialization defines the leaf entity categories of the tree (i.e., the subcategories) which are subsets of the root entity category. The arrows between the root entity category and the subset entity categories represent the pair of relationships is_a/includes depending on their directions.

Note that in Figure 3.6b the common attributes are shown at the supercategory level and not at the subcategory level. That is, the common attributes of the supercategory are inherited by the subcategories, and each subcategory is shown with only its specific attributes. For example, entity category 1 with four attributes in Figure 3.6a is transformed to subcategory 1 with only its specific attributes in Figure 3.6b. Figure 3.6c illustrates an example of generalization/specialization abstraction, where a supercategory BEAM has common attributes that are inherited by subcategories, such as WIDE FLANGE BEAM, PLATE GIRDER, and so on.

A supercategory or a subcategory has a corresponding class if the entities of the category are described by a single concept. In this case, a class is defined for the category as shown in Figure 3.7. As defined in this report, classes are different from categories. The classes shown in Figure 3.7 do not use inheritance because there is no inheritance between classes; a class must explicitly define each of the attributes of its entities. As shown in Figure 3.7 a category and its corresponding class have the relationship corresponds_to.

3.4.3 Aggregation and Decomposition

An aggregation abstraction defines a new compound category from a set of lower level entity categories (called simple categories). The term compound means that two or more entity categories are assembled to define a new entity category. Each entity of the compound category is composed of entities from the related simple categories. Aggregation is the process of assembling the compound category. Each entity of a lower level entity category is only a part of an entity of the compound category. The opposite process of aggregation is decomposition. Decomposition is a process of breaking a compound category into related lower level simple categories.

Figure 3.8a shows one entity category (entity category C) that has a large number of attributes, and several other entity categories (entity category 1, entity category 2, etc.) each of which has some of

the attributes of the entity category C. Application of aggregation abstraction to the entity categories shown in Figure 3.8a results in a single level aggregation/decomposition tree, as shown in Figure 3.8b. Aggregation defines the root entity category (i.e., the compound category) and decomposition defines the leaf entity categories (i.e., the simple categories). The arrows between the compound category and the simple categories represent the pair of relationships is_part_of/has_part depending on their directions.

Since compound category C is now defined by aggregation, it no longer has the same attributes that the simple categories have. That is, the attributes of the compound category are now hidden in the simple categories and are available by way of the has_part/is_part_of relationships between the compound category and the simple categories. Figure 3.8c illustrates an example of aggregation/decomposition. In this figure, a compound category PLATE GIRDER is defined by aggregating simple categories, such as PLATE GIRDER MATERIAL, PLATE GIRDER GEOMETRY, and so on.

4. Elements of Product and Process Models

This chapter discusses the elements of product and process models for structural design in more detail and presents graphical conventions that help to convey the meaning of the information in these models. Most elements of both the product and process models are defined using the concepts of semantic data models. The elements of the models are *entities, relationships, attributes, classes, instances, supercategories, subcategories, abstract categories, compound categories, and simple categories.* Two prefixes are applied to these elements, namely, *object-type* (O-type) and *activity-type* (A-type), to define the elements of a product model and a process model respectively. The elements of a product model are O-type entities, O-type relationships, O-type attributes, O-type classes, O-type instances, O-type supercategories, O-type subcategories, O-type abstract categories, O-type compound categories, and O-type simple categories. Similarly, the elements of a process model are A-type entities, A-type relationships, A-type attributes, A-type-classes, A-type instances, A-type supercategories, A-type subcategories, A-type abstract categories, A-type compound categories, and A-type simple categories.

A model for structural design should satisfy the following requirements:

1. The model should describe a full range of concepts used by structural designers.

2. The model should be consistent, simple, and easy to understand and use. A consistent notational convention should encourage the user to formalize ideas in a uniform way.

3. The model should be independent of any implementation model or programming language.

4. The model should be expressed in terms of specified graphical conventions.

The models in this report are developed considering these general requirements. The chapter is organized as follows. First, Section 4.1 discusses individually each element of the models. Then, graphical conventions for representing the models are discussed in Section 4.2.

4.1 Elements and Additional Concepts

As discussed earlier in Chapter 3, certain elements including entities, relationships, and attributes, are the primary components of semantic models. Other elements, including classes, instances, supercategories, subcategories, compound categories and simple categories are obtained by applying the abstraction mechanisms discussed in Section 3.5 to various categories of entities. Some additional concepts included in models for structural design, such as base attributes, derived attributes, value type specialization, value set specialization, and abstract categories, enable the development of more comprehensive and useful models. The discussion in this section includes all of these concepts.

4.1.1 Entity

Entities represent real-world things that are distinctly defined. Beams, columns, and frames are examples of entities for a building structure. Each entity has several attributes and corresponding values. Entities are generally grouped into entity categories according to their attributes. An entity category is a set of entities that are naturally grouped together because of their similar properties. Each entity category in a generalization model has a set of common attributes. These attributes characterize the entity category.

There are two basic types of entity categories: *O-type* entity categories for product models; and *A-type* entity categories for process models. O-type entity categories describe information about the objects being designed. A-type entity categories describe design activities and show how these activities use and generate information about the objects being designed. Supercategories, subcategories, classes, instances, compound categories, and simple categories of these two basic types of entity categories are obtained by applying the abstraction mechanisms discussed in Section 3.5.

4.1.2 Relationship

Relationships represent associations of two or more entity categories. The cardinality of a relationship category describes the number of entities in the categories that are involved in the

relationship; relationships may be either single valued or multi valued. As shown in Figure 4.1, if the relationship mapping (from left to right) is either one-to-one (1:1) or many-to-one (N:1), then the cardinality is "single valued" (SV) and if the relationship mapping is either one-to-many (1:M) or many-to-many (N:M), then the cardinality is "multi valued" (MV) (Thompson 1989). Relationships may also have attributes.

4.1.3 Attribute

Attributes represent elementary properties of entities or relationships. An entity is defined by its attributes. For example, the attributes of a line segment could be the coordinates of its two end points, as shown in Figure 4.2a. There are two basic types of attributes: *O-type attributes* for O-type entities; and *A-type attributes* for A-type entities. O-type attributes are used to describe properties of objects. A-type attributes are used to describe properties of activities. These attributes are characterized by value types, value sets, and cardinalities.

4.1.3.1 Value Types for Object-Type Attributes

O-type attributes are classified into two main types: (1) "data-valued" attributes (DVAs) which are attributes that have alphanumeric values that can be printed (Thompson 1989); and (2) "entity-valued" attributes (EVAs) which are attributes that refer to other entities. An O-type EVA is called an OEVA to distinguish it from an A-type EVA. The value of a DVA is printable, but the value of an OEVA is not, since it is a reference to another entity. The attributes shown in Figure 4.2a are data-valued attributes. The entity named line has four attributes which represent the coordinates of the two end points of a line segment in two dimensional space. In this case, the attributes have printable values (i.e., numbers). They are data-valued attributes.

The attributes shown in Figure 4.2b are entity-valued attributes. In this example, the information about a line, shown in Figure 4.2a, has been represented in a different way. The line entity has two OEVAs (i.e., starting_point and ending_point). The attributes starting_point and ending_point are not printable data but are entities. The two OEVAs describe the line while hiding the detailed information about the points. The detailed information about the points is then represented by way of other entities named point1 and point2, as shown in Figure 4.2c. The value of the OEVA

starting_point is the point1 entity which has two attributes that are the coordinates of the starting point of the line. Similarly, the value of the OEVA ending_point is the point2 entity which has two attributes that are the coordinates of the ending point.

The comparison of two representations (Figure 4.2a vs. Figures 4.2b and 4.2c) illustrates two important advantages that can be obtained by the use of EVA. When a line is first conceived, it is considered to be defined by two points, without concern for the detailed locations of the two points. However, it is difficult to determine that a line is defined by two points from Figure 4.2a. In contrast, Figure 4.2b indicates clearly that a line entity is composed of two points. Therefore, the use of EVAs enables information to be represented in more natural way and simplifies the representation of the LINE entity by reducing the number of attributes.

In addition to the two main types of O-type attributes (i.e., DVAs and OEVAs), O-type attributes are further classified into: (1) *base attributes*, which identify the entity uniquely and have values that are independent of other information in the model; and (2) *derived attributes*, which have values that are determined from the other information in the model (Hammer and McLeod 1981; Hull and King 1987; Rumbaugh 1991). An entity may have both base attributes and derived attributes. For example, the line entity has two attributes, point1 and point2. These attributes are base attributes and they are sufficient to represent the line entity. However, the length of the line can also be an attribute of the line entity. The value of this derived attribute can be calculated by using the information about the two points. One reason to include derived attributes (i.e., derived information) in a model is to make the model more realistic and useful than one composed of base attributes only.

Derived attributes are defined internally or externally. Internally derived (DI) attributes are derived using only the other attributes of the same entity. Externally derived (DE) attributes are derived using attributes of other entities. In some cases, a base or derived O-type attribute is not fully defined in a category, but it is intended for the attribute to be fully defined in the subcategories that inherit it. A subcategory-defined base (BS) attribute is a base attribute that is fully defined by subcategories that inherit it. A subcategory-defined derived (DS) attribute is a derived attribute for which the derivation procedure is defined in the subcategories that inherit it. Figure 4.3 summarizes the types of O-type attributes.

Subcategories in a generalization/specialization tree are usually specialized by adding more specific

attributes to the attributes that are inherited from a supercategory (Figure 3.6) as discussed in Chapter 3. One additional way to specialize subcategories is by changing the value type from subcategory-defined derived (DS) to either internally derived (DI) or externally derived (DE). This type of specialization is called "value type specialization", and is valid for DS attributes only. Figure 4.4 shows how subcategories are created by value type specialization; the subcategories define the derivation procedure for a DS attribute inherited from a supercategory. The subcategories must define the attribute to be either internally derived (DI) or externally derived (DE), and must provide a specific derivation procedure.

4.1.3.2 Value Types for Activity-Type Attributes

A-type attributes are classified into two main types: (1) "action-valued" attributes (AVAs) which are attributes whose values are actions; and (2) "entity-valued" attributes (EVAs) which are attributes that refer to other entities. An A-type EVA is called an AEVA. The value of an AVA is executable, which means the design activity represented by the AVA is implemented by operations and therefore is not decomposed into more detailed design activities. In contrast, the value of an AEVA is not executable, since it is a reference to another entity. The attributes shown in Figure 4.5a are action-valued attributes. The beam elaboration entity has six attributes which represent design tasks for calculating bending moments and shear forces. In this case, the attributes have executable values since the design tasks they describe do not require further decomposition into more detailed tasks. They are action-valued attributes.

The attributes shown in Figure 4.5b are entity-valued attributes. In this example, the design tasks for beam elaboration, shown in Figure 4.5a, are represented in a different way. The beam elaboration entity has two AEVAs (i.e., calculate_moment_diagram and calculate_shear_force_diagram). The attributes calculate_moment_diagram and calculate_shear_force_diagram are not executable actions but are references to other entities. The two AEVAs describe the major design tasks for beam elaboration while hiding the more detailed design tasks of calculating bending moments and shear forces. These more detailed design tasks are then represented by way of other entities named bending moment diagram calculation and shear force diagram calculation, as shown in Figure 4.5c. The bending moment diagram calculation entity has three attributes whose values are the actions for calculating the bending moment diagram,

and the shear force diagram calculation entity has three attributes whose values are the actions for calculating the shear force diagram.

The comparison of two representations (Figure 4.5a vs. Figures 4.5b and 4.5c) illustrates the advantages of using AEVAs. That is, the use of EVAs for A-type attributes enables design activities to be represented in a natural way from the most abstract tasks to the most detailed tasks, and simplifies the representation of A-type entities by reducing the number of attributes.

In addition to the two main types of A-type attributes (i.e., AVAs and AEVAs), A-type attributes are also classified into: (1) *base attributes*, which have values that are independent of other design tasks in the model; and (2) *derived attributes*, which have values that are dependent on the other design tasks in the model. A derived A-type attribute (i.e., a derived design task) must have a derivation procedure to determine the correct design task to be carried out. Similar to O-type derived attributes, A-type derived attributes are also defined either internally or externally. Internally derived (DI) attributes are dependent only on the other design tasks of the same entity. Externally derived (DE) attributes are dependent on design tasks of other entities. If a base or derived A-type attribute is not fully defined in a category, but is intended to be defined in the subcategories that inherit it, it is a subcategory-defined attribute. Subcategory-defined base (BS) attributes, and subcategory-defined derived (DS) attributes of A-type entities are similar in concept to those of O-type entities. Figure 4.6 summarizes the types of A-type attributes

Note that value type specialization is also valid for A-type subcategories. That is, A-type subcategories may be created by changing a value type for a DS attribute inherited from a supercategory. The subcategories must define the attribute to be either internally derived (DI) or externally derived (DE), and must provide a specific derivation procedure.

4.1.3.3 Value Sets for Object-Type Attributes

Each O-type attribute has a value set. That is, each O-type attribute is associated with a set of possible activities for that attribute. The value set is part of the definition of the O-type attribute. The value set of an O-type attribute is either a data category or an entity category depending on the value type of the attribute. If the attribute is a DVA, the value set is a data category (e.g., the set of real numbers). If the attribute is an OEVA, the value set is an entity category. Thompson (1989)

calls the value set of an entity-valued attribute the "range class" of the attribute. In this report, the range class is called the "range category", because the term "class" used by Thompson (1989) is similar to term "entity category" of this report.

As noted earlier, subcategories in a generalization/specialization tree are usually specialized by adding more specific attributes. Another way to specialize categories, value type specialization, was defined earlier. A third way to specialize categories is by defining different value sets for inherited attributes. This type of specialization is called "value set specialization" and is valid for EVAs only. Figure 4.7 shows how subcategories are created by value set specialization; the subcategories redefine the value sets for the attributes inherited from the supercategory. A BS-DVA or a BS-OEVA (Figure 4.3) defined in a supercategory is an attribute that requires the subcategories to redefine the value set through value set specialization. Value set specialization allows the subcategories in a generalization/specialization tree to have similar attributes (which differ in their value sets and/or value types). The application of value set specialization is discussed in more detail in Section 5.4.3.

4.1.3.4 Value Sets for Activity-Type Attributes

Each A-type attribute also has a value set. That is, each A-type attribute is associated with a set of possible operations for that attribute. The value set is part of the definition of the A-type attribute. The value set of an A-type attribute is either an action category or an entity category depending on the value type of the attribute. If the attribute is an AVA, the value set is an action category and if the attribute is an AEVA, the value set is an entity category. Figure 4.8 shows the description of an action category which consists of an action category name, an input, an output and expressions. The action category name must be identical to the name of a corresponding value set. The input provides information that is used or transformed by the expressions. The output includes an information that is generated through the expressions. Expressions include simple equations needed to use and generate the input and output. Figure 4.9 shows the descriptions of several action categories defined for the value sets of plate girder design tasks.

Note that value set specialization is also valid for A-type subcategories for the same purpose as for O-type attributes. That is, A-type subcategories may be created by changing a value set for an attribute inherited from a supercategory.

4.1.3.5 Cardinality

An attribute (i.e., O-type or A-type attribute) is characterized by a cardinality that indicates the number of values that may apply at one time to the attribute. There are two types of attribute cardinalities: single valued (SV) and multi valued (MV). An O-type multi-valued attribute can have more than one value at one time. For example, if a beam entity has an attribute "load", and it has values for dead load, live load, and so on, then this attribute is multi- valued. An A-type multi-valued attribute means that the design activity is used more than once (i.e., iterated) within a higher level design activity. For example, if the A-type entity BEAM DESIGN PROPOSAL has an attribute size_member, and the values of this attribute are iterations of the size_member activity to determine an appropriate member size, then this attribute is multi-valued.

4.1.4 Class

The root of a classification/instantiation abstraction tree is a class. A class is a template for creating entities that are grouped into an entity category. As noted earlier, a class is different from an entity category because a class explicitly defines each attribute while an entity category may use inheritance to enable certain attributes to be defined in a supercategory. Thus a class provides the prototypical format for all the attributes of a set of entities, with the corresponding value types, value sets, and cardinalities. The detailed format for a class is discussed in Section 4.2. An *O-type class* is a template for generating O-type instances that describe information about objects being designed. An *A-type class* is a template for generating A-type entities that describe design activities.

4.1.5 Instance

The leaves of a classification/instantiation abstraction tree are instances which are individual entities from the predefined class. The term "instance" will be used to refer to an entity generated from a specific class. An instance is an occurrence of a class for which values of the attributes can be specified. *O-type instances* describe information about objects being designed. *A-type instances* describe design activities.

4.1.6 Supercategory

A supercategory is the root entity category of a single level generalization/specialization tree. A supercategory summarizes the attributes common to the lower level entity categories in the single level generalization hierarchy (i.e., the subcategories). All the attributes of a supercategory are inherited by the subcategories. If generalization abstraction is applied to a set of O-type entity categories, the root of the resulting single level generalization/specialization tree is an *O-type supercategory*. If generalization abstraction is applied to a set of A-type entity categories, the root of the resulting single level generalization/specialization tree is an *A-type supercategory*.

4.1.7 Subcategory

Subcategories are the leaves of a single level generalization/specialization tree. Subcategories have the attributes inherited from a supercategory as well as other attributes that are independent of the other subcategories. As discussed in Section 4.1.3, subcategories also can be created by value set specialization of attributes inherited from the supercategory. *O-type subcategories* and *A-type subcategories* can be made from their corresponding supercategories.

4.1.8 Abstract Category

A category with no corresponding class is called an abstract category. An abstract category has no instances, although the classes corresponding to its subcategories may have instances, as shown in Figure 4.10a. The term "abstract category" is motivated by the concept of "abstract classes" that is used in object-oriented design (LaLonde and Pugh 1990; Rumbaugh 1991). An abstract category is used to define common derived attributes for a set of subcategories. Each subcategory may have different base attributes, and the procedures for determining the values of the common derived attributes are defined independently within the subcategories.

Abstract categories and generalization abstraction can be utilized to manage different categories of entities uniformly. For example, if several beam design alternatives, such as a wide flange beam, a plate girder, and an open web bar joist, are proposed as solutions to a beam design problem, they should be compared with each other in an unbiased way. The properties of these alternative beams

are different. However, in order to compare them, several general attributes, such as beam_weight, can be defined for these different types of beams. The result of this generalization abstraction is shown in Figure 4.10b. The supercategory BEAM is an abstract category with no instances because it does not provide enough information to calculate the beam weight. Instead, the supercategory defines the common derived attribute beam_weight whose value is calculated within the subcategories (i.e., WIDE FLANGE BEAM, PLATE GIRDER, and OPEN WEB BAR JOIST). Each subcategory inherits the attribute beam_weight and defines the calculation of beam_weight from the attributes for that subcategory.

4.1.9 Compound Category

A compound category is the root entity category of a single level aggregation/decomposition tree. A compound category is defined by aggregating a set of lower level related simple categories. If aggregation abstraction is applied to a set of O-type entity categories, the root of the resulting single level aggregation/decomposition tree is an *O-type compound category* And if aggregation abstraction is applied to a set of A-type entity categories, the root of the resulting single level aggregation/decomposition tree is an *A-type compound category*

4.1.10 Simple Category

Simple categories are the leaves of a single level aggregation/decomposition tree. A simple category has a set of attributes that represent a part of a compound category. *O-type simple categories* and *A-type simple categories* can be made by decomposing the corresponding compound O-type and A-type categories into lower level entity categories.

4.2 Graphical Representation

The graphical conventions for semantic models should be capable of representing the concepts defined in the previous section. Several properties of graphical representations are discussed by Thompson (1989). The following requirements of a graphical representation defined based on those given by Thomson (1989):

1. The concept of an entity category.

2. The relationships among categories (e.g., supercategory-subcategory, etc.).

3. The attributes of an entity category.

4. The cardinality of an attribute.

5. The value type of an attribute (e.g., B-OEVA, DE-DVA,etc.).

6. The value set of an attribute.

7. The instances of a class.

Three kinds of graphical representations that satisfy these requirements are discussed in this section. They are the modified E-R diagram, the table representation, and the modified semantic data model (SDM) diagram. The modified E-R diagram is developed from the E-R diagram (Chen 1976). This model is convenient to show the relationships among entities. The table representation is developed to show details about the attributes of an entity category, such as the value type, value set, and cardinality of each attribute. The modified SDM diagram is developed from the SDM diagram (Thompson 1989). The modified SDM diagram clarifies and simplifies the SDM diagram. These graphical representations are compatible and interchangeable. That is, for example, the table representation can be easily converted to the modified SDM diagram, and the modified SDM diagram can be converted to a table representation.

4.2.1 Modified E-R Diagram

As discussed earlier, the E-R diagram represents entity categories and relationship categories in terms of rectangles and diamonds respectively. The cardinality of the relationship is indicated along with the relationship, as in Figure 3.1. The graphical representation for an attribute, a bar with a small circle at the end, was previously added to the original E-R diagram (Batini 1992). However, the E-R diagram has several weakness: (1) a class is not distinguished from its instances; (2) the value set and value type of an attribute are not defined; and (3) hierarchical relationships, such as those established by using the abstraction mechanisms described in Section 3.4, are not emphasized.

The modified E-R diagram, developed in this report, includes additional aspects that are not included in the E-R diagram (Figure 4.11a). This modified E-R diagram distinguishes classes from their instances; classes (and categories) are represented by rectangles and instances are represented by rectangles with rounded corners. The modified E-R diagram organizes the entities in hierarchies. This report uses three types of hierarchical relationships (i.e., member_of/prototype_of, is_a/includes, and is_part_of/has_part) obtained by applying the three pairs of abstraction mechanisms (i.e., classification/instantiation, generalization/specialization, and aggregation/decomposition). The value set of an attribute is represented in square ([ ]) brackets, next to the attribute bar. In the representation of an instance, the value set is replaced by the value. A value type is represented in parentheses (( )) shown below the attribute bar.

The cardinality of a relationship between entity categories is indicated along with the relationship, as in the E-R model. However, when the relationship mapping between two entity categories is one-to-one, the diamond shape normally used to represent the relationship is deleted. Instead, the relationship is written next to the line between the two entity categories. Figure 4.11a shows the case when relationship mapping between two entity categories is one-to-one. The cardinality of an attribute is described as follows: if an attribute is single valued, then the attribute bar ends with an empty circle, and if an attribute is multi valued, then the attribute bar ends with a black circle. Figure 4.11b shows two entity categories using a modified E-R diagram to represent the information about a line and a point respectively. The figure uses the has_parts relationship which has cardinality of "1 to 2" between the LINE and POINT entity categories.

4.2.2 Table Representation

In order to represent the details of the attributes of an entity category, the table representation has been developed in this report (e.g., Figure 4.12). The table representation is similar to the modified E-R model. However, the table representation uses a table format for the attribute information and does not explicitly show the relationships between entity categories. Figure 4.12a shows the table representation for an entity category, which includes the entity category name, the supercategory name, and the attribute names, value sets, value types and cardinalities of the attributes. The entity category name identifies the collection of entities. The supercategory name indicates the relationship between the category and its supercategory. This replaces the is_a

relationship of the modified E-R diagram. If a supercategory name is "none", then this category is the root category of a generalization/specialization tree.

The table representation for an instance is similar to that of an entity category, as shown in Figure 4.12b. The table representation for an instance includes the instance name, its class name, and the attribute names, values (rather than value sets), value types and cardinalities of the attributes. The class name replaces the member_of relationship of the modified E-R diagram.

Figure 4.12c shows an example of the table representation for a line and a point. It is assumed that these entity categories are root categories in the model.

4.2.3 Modified SDM Diagram

SDM diagrams were developed to support the semantic data model (Thompson 1989). Figure 4.13a shows the SDM diagram convention. As noted earlier, a "class" in Thompson's terminology corresponds to an "entity category" in this report. An entity category is indicated by a vertical line connected to a horizontal line at the top. The name of the entity category is written in angled brackets (< >) above the horizontal line. The attributes of the entity category are indicated by additional horizontal lines attached to the vertical line. Each additional horizontal line is labeled with the name of the attribute. The attribute line terminates with the name of the category to which the attribute maps (i.e., its value set). The value set, which refers to an entity category, a data category, or an action category, is written in angled brackets (< >). The attribute line terminates in a single arrow if the attribute is single valued, in a double arrow if the attribute is multi valued. The supercategory-subcategory relationship (i.e., superclass-subclass relationship in Thomson's terminology) is represented by indenting one entity category description within another. Subcategories are indicated by placing the vertical line for the subcategory to the left of the vertical line for its supercategory. One drawback of the SDM diagram is that an instance of a class (corresponding to an entity category) is not clearly distinguished from its class. However, an instance usually includes values instead of value sets for the attributes. An example of an instance is shown in Figure 4.13b.

The value type of an attribute is not clearly defined by the SDM diagram. Thus, in order to represent an EVA, the entity category that corresponds to the value set of an attribute is included in

the SDM diagram. This is illustrated in Figure 4.13c. In this case, the name of the value set is replaced by a vertical line which connects the attribute to the entity category that corresponds to the value set of the attribute. This category is called the "range category" (i.e., the range class in Thompson's terminology). In the SDM diagram the range category is given additional attributes that describe the relationship between the range category and entity category with the corresponding EVA. For example, two attributes of the POINT entity category in Figure 4.13c, named is_end_of and is_start_of, represent the relationships between the LINE entity category and the POINT entity category. These attributes of the POINT entity category are optional because (1) they are not essential to describe the POINT entity category and (2) the relationships between the LINE and POINT entity categories can be represented by the attributes in the LINE entity category. When these attributes are included in a model, it is difficult to achieve consistent representation. For example, if the POINT entity category is also used as a range category for attributes of a TRIANGLE entity category, as in Figure 4.13d, the is_end_of and is_start_of must be replaced by other attributes such as is_vertex1_of, is_vertex2_of, and is_vertex3_of. That means the attributes of any category that is a range category must be changed as needed to describe its relationships to other categories.

The modified SDM diagram convention used in this report varies from that used by Thompson (1989) as follows (Figure 4.14a): (1) a value set is written in square brackets ([ ]), to distinguish it from an entity category, which is written in angled brackets (< >); and (2) the value type is written in parentheses (( )) under the attribute line. Figure 4.14a shows a modified SDM diagram. The instance representation of the modified SDM diagram is given in Figure 4.14b. Note that "subcategory C" has been changed to "instance of class C" and all the attributes of the instance (attributes 1, 2, 5 and 6) are shown explicitly in Figure 4.14b, because instances are derived from a class by way of the member_of relationship and the class defines every attribute explicitly unlike the category which may have inherited attributes. Two examples of the modified SDM diagram are given in Figure 4.14c and Figure 4.14d, which simplify the SDM diagrams shown in Figures 4.13c and 4.13d. Note that the POINT entity category is the same in both Figures 4.14c and 4.14d unlike Figures 4.13c and 4.13d, because new attributes are not added to a range category (e.g., POINT) in the modified SDM diagram.

5. Models for Structural Design

The purposes of this chapter are to present an approach for using the product and process model, described in the previous chapter, to develop product and process models for structural design, and to present an approach to integrate the product and process models. The chapter focuses on models for the development and comparison of several design alternatives that are proposed as solutions to a given design problem. The requirements for these models are identified, and information modeling strategies that address these requirements are then developed. In order to simplify the discussion, the following limitations are imposed: (1) the design alternatives for only one design problem are considered; (2) the design of a structural component (e.g., design of a beam) is the only type of design problem considered; and (3) the entities needed to describe design alternatives and to develop these alternatives are the only entities considered. That is, in this chapter, O-type entities are used to represent design alternatives; and A-type entities are used to represent the development tasks of the MSD model. Other O-type or A-type entities are not considered.

The chapter is organized as follows. First, Section 5.1 examines the development and comparison of design alternatives to establish the requirements for the design product and process models. These requirements are summarized in Section 5.2. Section 5.3 discusses the proposed approach for developing product and process models. Then, several information modeling strategies that address the product and process model requirements are discussed in Section 5.4. Section 5.5 presents a general procedure for the development of product and process models. Section 5.6 discusses integration of the product and process models developed in Section 5.5. Finally, instance models are discussed in Section 5.7.

5.1 Characteristics of Structural Design Alternatives

When structural design is viewed as a process of problem solving, several competing design alternatives may be considered as possible solutions to a design problem. During the problem solving process, less desirable design alternatives are eliminated. The remaining design

alternatives are refined through the several design stages (e.g., conceptual design, preliminary design, and detailed design) in which more detailed information about each remaining design alternative is developed. During the different design stages, a design alternative evolves toward a more complete solution. The evolution of one design alternative is independent of the other design alternatives. Thus, information about each design alternative can be managed independently.

Careful examination of the process of developing and comparing several design alternatives reveals three important properties: (1) design alternatives are *plural,* because it is difficult to identify directly the best solution to a design problem, and several design alternatives may be considered as possible solutions; (2) information about each design alternative is *independent* of information about other alternatives, however, a set of design alternatives must have some common properties to form the basis for comparison and ranking; and (3) the information about a design alternative *grows* as the design process proceeds from conceptual to detailed design. Figure 5.1 shows several different design alternatives with different sets of design properties, and Figure 5.2 illustrates how the information grows for each design alternative, as additional design properties are considered.

5.2 Requirements for Product and Process Models

Certain requirements for product and process models that will support the development and comparison of design alternatives can be established as follows, based on the observations in the previous section.

5.2.1 Requirements for Product Model

1. O-type categories should be defined for all entities considered in the design process; in particular one O-type category should be defined for each type of design alternative considered.

2. The O-type category for a design alternative should define a unique set of attributes to represent the design properties for the alternative.

3. Partial descriptions of design alternatives should be allowed, and the consideration of additional design properties should be permitted to allow the evolution of design alternatives.

4. O-type categories for the different design alternatives for a given design problem should have some common attributes to allow unbiased comparison of the alternatives.

5. O-type categories should define attributes for all design properties that are used and generated by the corresponding activities in the process model.

5.2.2 Requirements for Process Model

1. A-type categories should be defined for all activities needed in the design process; in particular one A-type category should be defined for the design activities for each type of design alternative considered.

2. The A-type category for a design alternative should define a unique set of attributes to represent the design tasks for the alternative.

3. A-type categories should define attributes that represent design tasks of increasing number and complexity.

4. A-type categories for design alternatives should have sufficient uniformity to be managed in a consistent way.

5. Process model activities should be defined in terms of operations on product model entities.

The requirements for a process model are similar to those of a product model. The product model and process model are related because both models are needed to describe information about the design of a structural system or component.

5.3 Proposed Approach

The information related to the design of a specific structural system or component is represented by two formal models: (1) a product instance model; and (2) a process instance model. As discussed in Section 2.2, these models are created using instances from: (1) a product generalization model; and (2) a process generalization model. The product generalization and process generalization models are usually considered separately. However, in this chapter, product and process models

are developed together because: (1) there is a *similarity* in the requirements of product and process models; and (2) there is a need to *integrate* the product and process models. The similarity between requirements for product and process models means that once the categories and relationships of a product model are defined, similar categories and relationships for a process model are more easily defined using the same information modeling concepts (i.e., abstraction mechanisms, the concepts of entities, relationship, etc.). Integration of product and process models means that each O-type entity describes information created and used by the corresponding A-type entity (i.e., a design activity), and each A-type entity is described as operations on the corresponding O-type entity. Thus, the O-type and A-type entities function as the medium for integration of the corresponding models. Developing the product and process models together enhances the integration of the models, allowing partially developed models to be checked to determine if the product model directly supports the process model, and to determine if the process model is expressed in terms of the product model.

5.4 Information Modeling Strategy

In order to develop product and process models that address the requirements discussed in Section 5.2, an information modeling strategy consisting of the following four elements is employed: (1) the use of generalization of design alternatives toward a uniform representation; (2) the use of entity-valued attributes (EVAs) to condense detailed information into single attributes; (3) the use of the value set specialization to change the value set of an EVA as more specialized entity categories are defined; and (4) the use of a Primitive-Composite (P-C) type approach.

5.4.1 Use of Generalization for Uniform Representation

The design alternatives for one design problem are initially considered to be equally likely solutions to the problem. For example, at certain points in the design process, each design alternative for a beam design problem may be considered to be "the beam" without concern for how the alternative's physical properties differ from those of the other alternatives. In particular, this occurs when the design alternatives are compared to determine the best alternative for the design problem. Therefore, it is desirable to provide a "uniform representation" of beam design

alternatives in the product model. Uniform representation requires the definition of an entity category with attributes that are general enough to represent common properties of a set of different types of design alternatives. The uniform representation allows structural designers to develop and compare different design alternatives in a consistent way.

In order to define an entity category that describes different types of design alternatives uniformly, generalization abstraction is used to identify the common properties of the different design alternatives. The generalization abstraction results in a generalization/specialization tree that consists of a supercategory with attributes describing the common properties of a set of design alternatives, and subcategories for the design alternatives.

Figure 5.3a shows an example of a product model for beam design alternatives based on generalization abstraction. The O-type supercategory BEAM defines attributes for the common properties, such as weight, cost, material_density, elastic_modulus, section_area, moment_of_inertia, and so on, without considering more specific properties of the different beam design alternatives.

Figure 5.3b shows an example of a process model for the development of beam design alternatives based on generalization abstraction. The A-type supercategory BEAM DESIGN defines attributes for the common tasks, such as calculate_moment_diagrams, calculate_shear_force_diagrams, size_member, review_member_for_moment, and so on, without considering more specific design tasks for the different beam design alternatives.

5.4.2 Use of Entity-Valued Attributes

In the previous section, the attributes of the BEAM category describe the weight, cost, material, and geometric properties of a beam. However, the representation of the BEAM category in Figure 5.3a is more complicated than is desirable. A simplified description can be obtained by redefining the common properties in terms of EVAs. The use of EVAs for a product model (i.e., OEVAs) has two benefits: (1) a simplified and clearer description of a structural system or component can be obtained by hiding detailed properties within OEVAs; and (2) several related properties can be grouped together (condensed) into a single OEVA which serves as an abstract description for the related properties. Figure 5.4a shows an example of a product model for beam design alternatives,

in which the BEAM category is defined in terms of material, geometry, and merit OEVAs that condense three sets of related properties into three OEVAs. A beam is initially conceived as having a material, a geometry, and a merit as its components. The more detailed properties are described in the range categories that correspond to the value sets of these OEVAs. That is, each OEVA has a value set that refers to the range category with attributes that describe more detailed properties. The material OEVA has a value set that refers to the BEAM MATERIAL category. This entity category has attributes to describe detailed material properties, such as material_density and elastic_modulus. Similarly the geometry OEVA has a value set that refers to the BEAM GEOMETRY category. This category has detailed attributes to describe geometric properties, such as section_area and moment_of_inertia. The merit OEVA has a value set that refers to the BEAM MERIT category. This category has attributes to describe various measures of merit for selecting the best alternative, such as weight and cost.

The representation of the BEAM DESIGN category (Figure 5.3b) can also be simplified by using EVAs (i.e., AEVAs). The use of AEVAs for a process model has two benefits: (1) a simplified and clearer description of the design activities for a structural system or component can be obtained by hiding detailed design tasks within AEVAs; and (2) several related design tasks can be condensed into a single AEVA which serves as an abstract description for the related tasks. Figure 5.4b shows an example of a process model for beam design, which is described in terms of elaboration, design proposal, and review AEVAs. Beam design is initially conceived as consisting of elaboration, design proposal, and review tasks. The more detailed design tasks are described in range categories. Each AEVA has a value set that refers to the range category with attributes that describe more detailed design tasks. The elaboration AEVA has a value set that refers to the BEAM ELABORATION category. This entity category has attributes to describe detailed elaboration tasks, such as calculate_moment_diagrams and calculate_shear_force_diagrams. Similarly the design_proposal AEVA has a value set that refers to the BEAM DESIGN PROPOSAL category. This category has attributes describing detailed design proposal tasks, such as size_member.

5.4.3 Use of Value Set Specialization

In Section 5.4.1, generalization abstraction was used to develop supercategories that have common properties for a set of design alternatives. In Section 5.4.2, EVAs were used to simplify the supercategory, and to place related properties into entity categories (i.e., range categories) corresponding to the value sets of the EVAs. In this section, subcategories that describe different types of design alternatives are defined by value set specialization of the EVAs. The value set specialization process used here does not add attributes to each subcategory but defines different value sets for the inherited EVAs, as discussed in Section 4.1.

Figure 5.5a completes the generalization/specialization tree of Figure 5.4a by using value set specialization to define attributes for the subcategories of the BEAM category. As discussed earlier, the BEAM category has material, geometry, and merit OEVAs, with value sets of that refer to the range categories BEAM MATERIAL, BEAM GEOMETRY, and BEAM MERIT respectively (Figure 5.4a). The subcategories of the BEAM category have different value sets for the material, geometry, and merit OEVAs that refer to more specific range categories that are appropriate for the specific beam design alternatives. For example, the WIDE FLANGE BEAM subcategory has "steel", "wide flange beam geometry", and "wide flange beam merit" as value sets for the material, geometry, and merit OEVAs; the PLATE GIRDER subcategory has "steel", "plate girder geometry", and "plate girder merit" as value sets for the material, geometry, and merit OEVAs. The range categories of these more specific value sets have more specific attributes. For example, the PLATE GIRDER GEOMETRY category has specific attributes for plate girder geometry properties such as web_thickness, web_depth, flange_thickness, and so on.

Figure 5.5b completes the generalization/specialization tree of Figure 5.4b by using value set specialization to define attributes for the subcategories of the BEAM DESIGN category. The BEAM DESIGN category has elaboration, design proposal, and review AEVAs, with value sets that refer to the range categories BEAM ELABORATION, BEAM DESIGN PROPOSAL, and BEAM REVIEW respectively. The subcategories of the BEAM DESIGN category have different value sets for the elaboration, design_proposal, and review AEVAs that refer to more specific range categories. For example, the WIDE FLANGE BEAM DESIGN subcategory has "wide flange beam elaboration", "wide flange beam design proposal", and "wide flange beam review" as value sets for the elaboration, design_proposal, and review AEVAs; the PLATE GIRDER DESIGN subcategory has "plate girder

elaboration", "plate girder design proposal", and "plate girder review" as value sets for the elaboration, design_proposal, and review AEVAs. The range categories of these more specific value sets have more specific attributes for more specific design tasks. For example, the PLATE GIRDER DESIGN PROPOSAL category has specific attributes for plate girder design tasks, such as propose_web_thickness, propose_web_depth, propose_flange_thickness, and so on.

5.4.4 Use of a Primitive-Composite (P-C) Type Approach

A uniform representation of design alternatives is presented in Sections 5.4.1, 5.4.2, and 5.4.3. The representation uses entity categories with EVAs that describe common design properties (or tasks) for a set of design alternatives. Entity categories for different design alternatives are specialized by redefining the value sets for the EVAs. Here, the following points are noted:

1. The range categories corresponding to the various value sets defined for one EVA can be organized into a generalization/specialization hierarchy because of the similarities in their attributes. For example, the BEAM GEOMETRY, WIDE FLANGE BEAM GEOMETRY, and PLATE GIRDER GEOMETRY categories in Figure 5.5a are range categories of the geometry OEVA. All attributes of these categories describe geometric properties of beam design alternatives. The BEAM GEOMETRY category has section_area and moment_of_inertia attributes that are part of the geometric description of any beam design alternative. Thus, the WIDE FLANGE BEAM GEOMETRY and PLATE GIRDER GEOMETRY categories also include these attributes.

2. The generalization/specialization hierarchy for the range categories of one EVA is separate from the generalization/specialization hierarchies for the range categories of other EVAs, because each EVA represents a fundamental concept.

These points lead us to include two types of generalization hierarchies in the product and process models, based on the P-C approach discussed in Chapter 2: (1) *primitive generalization hierarchies* (primitive hierarchies), which consist of categories that describe atomic (or fundamental) concepts for a structural system or component; and (2) *composite generalization hierarchies* (composite hierarchies), which consist of categories that aggregate these atomic concepts to describe a structural system or component more completely. Primitive hierarchies that describe design properties are "O-type primitive hierarchies" and those that describe design tasks

are "A-type primitive hierarchies". Composite hierarchies that describe design properties are "O-type composite hierarchies" and those that describe design tasks are "A-type composite hierarchies". The entities of primitive hierarchies are "primitive entities" and those of composite hierarchies are "composite entities". Primitive entities of product models are "O-type primitive entities" and those of process models are "A-type primitive entities". Composite entities of product models are "O-type composite entities" and those of process models are "A-type composite entities". Note that the primitive and composite hierarchies are not generalization models. Rather, primitive and composite hierarchies together comprise a generalization model.

A primitive hierarchy is a homogeneous hierarchy that consists of only the range categories for one EVA of a composite hierarchy. All attributes of the categories in the primitive hierarchy are related to the EVA. The primitive hierarchy is created for an EVA if there are more than two value sets (with corresponding range categories) defined for the EVA, and if the range categories have common properties among them. A generalization/specialization process is used to develop the primitive hierarchies.

Figures 5.6 illustrates several O-type primitive hierarchies for a product model. The root categories of these hierarchies are abstract categories with subcategory-defined derived (DS) attributes that describe common design properties. For example, the section_area and moment_of_inertia attributes of the BEAM GEOMETRY category in Figure 5.6d are common to the PLATE GIRDER GEOMETRY, the WIDE FLANGE BEAM GEOMETRY, and the other subcategories. The section_area and moment_of_inertia attributes have the DS-DVA value type. These attributes are inherited by the subcategories, and each subcategory provides a procedure for deriving the values of these attributes from the specific attributes of the subcategory. For the PLATE GIRDER GEOMETRY subcategory, the specific attributes web_thickness, web_depth, flange_thickness, and so on, are used to determine values for the derived attributes (section_area and moment_of_inertia) inherited from the BEAM GEOMETRY category.

Figures 5.7 illustrates several A-type primitive hierarchies for a process model. The root categories of these hierarchies are abstract categories with subcategory-defined base (BS) attributes that describe common design tasks. For example, the size_member attribute of the BEAM DESIGN PROPOSAL category in Figure 5.7d is common to the WIDE FLANGE BEAM DESIGN PROPOSAL, the PLATE GIRDER DESIGN PROPOSAL, and the other subcategories. The size_member attribute is an

AEVA. For WIDE FLANGE BEAM DESIGN PROPOSAL, the value set for size_member refers to a range category WF BEAM SIZE MEMBER, which has attributes for the specific actions needed to size a wide flange beam. For PLATE GIRDER DESIGN PROPOSAL, the range category for size_member, PLATE GIRDER SIZE MEMBER, has attributes for the specific actions needed to size a plate girder such as a web thickness, a web depth, a flange thickness, and a flange width.

Note that a primitive hierarchy is independent or dependent. A primitive hierarchy is independent when the attributes of the range categories of one EVA in the composite hierarchy (i.e., the entities of the categories in the primitive hierarchy) describe the fundamental concept corresponding to the EVA without involving attributes of the range categories of other EVAs. For example, the attributes of the BEAM MATERIAL category (Figure 5.4a) such as material_density, elastic_modulus, and so on, can describe the material properties of beams independently, and similarly the attributes of the BEAM GEOMETRY category, such as section_area, moment_of inertia, and so on, can describe the geometry properties of beams independently. Therefore, these primitive hierarchies are independent.

A primitive hierarchy is dependent when the attributes of the range categories of one EVA in the composite hierarchy describe the fundamental concept corresponding to the EVA using the attributes of the range categories of other EVAs in the composite hierarchy. For example, the attribute weight of the BEAM MERIT category (Figure 5.4a) depends on information about geometry and material properties because the calculation of the weight requires information such as beam length, section area, and material density. Therefore, the beam merit primitive hierarchy is dependent.

A composite hierarchy consists of entity categories which describe design information or design activities for the design alternatives for a structural system or component. As shown in Figures 5.8 and 5.9, the entities in a composite hierarchy are defined by aggregating two or more primitive entities. Aggregation uses the is_part_of relationship, as discussed in Section 3.4 (i.e., composite entities are compound entities). Each composite entity category is made by aggregating range categories corresponding to the value sets of its EVAs. For example, in Figure 5.8 the O-type composite category WIDE FLANGE BEAM is made by aggregating the range categories, STEEL, WIDE FLANGE BEAM GEOMETRY and WIDE FLANGE BEAM MERIT. One advantage of a P-C type approach is achieved by using the primitive category STEEL in the definition of both the WIDE

FLANGE BEAM category and the PLATE GIRDER category. This allows reuse of existing entity definitions. In Figure 5.9, the A-type composite category WIDE FLANGE BEAM DESIGN is made by aggregating the range categories, WIDE FLANGE BEAM ELABORATION, WIDE FLANGE BEAM DESIGN PROPOSAL, and WIDE FLANGE BEAM REVIEW.

5.5 Product and Process Modeling Procedure

As proposed earlier in Section 5.3, product and process generalization models are developed together in this research. This involves defining entities, attributes, and the relationships between the entities. The development of models in this report is based on the following approach: (1) entities are defined using a top-down approach; (2) the entities are organized using the information modeling strategy which was discussed in the previous section; and (3) models are refined through iteration. Note that the top-down approach for defining entities means that composite entities are first defined. Then primitive entities are defined for the parts (i.e., the primitive concepts) of the composite entities. This sequence is used, because it is easier to begin the model by describing things that are considered in the design process rather than the parts of these things. The procedure described here can be applied equally to either the development of a product model or the development of a process model. The procedure is as follows:

1. Build a composite generalization hierarchy that provides entity categories for the design alternatives considered, such as wide flange beam, plate girder, and so on. For uniform representation of design alternatives, use generalization abstraction to identify a supercategory for these entity categories, as in Figure 5.10.

2. Refine the supercategory using EVAs, as shown in Figure 5.11. When each EVA has been identified, related common properties are grouped into a set of attributes for the range category corresponding to the value set of the EVA. This process is equivalent to a decomposition of the supercategory into identified parts as shown in Figure 5.12. Note that in Figure 5.11, value sets are defined only for the composite supercategory. The range categories for these value sets become the supercategories of the primitive hierarchies, and thus these value sets define the primitive concepts considered in the model. Value sets for the composite subcategories are defined after primitive hierarchies are complete.

3. Build primitive generalization hierarchies separately for the range categories of the EVAs identified in the second step. The primitive hierarchies should include subcategories needed to provide the parts (i.e., the primitive concepts) of the composite subcategories. Figure 5.13 shows several primitive hierarchies. The supercategory of each primitive hierarchy is the range category for one EVA of the supercategory of the composite hierarchy. For example, the supercategory of primitive hierarchy 1 in Figure 5.13a is the range category for EVA1 that was identified in Figure 5.11, and thus it has attributes for common properties 1 and 2. Primitive subcategories are defined through either a specialization process that adds more specific attributes; a value type specialization process that redefines value types for inherited attributes; or a value set specialization process that redefines value sets for inherited attributes, as shown in Figure 5.13.

4. Complete the composite hierarchy by using value set specialization to add EVAs to the subcategories of the composite hierarchy (Figure 5.11) as shown in Figure 5.14. The composite and associated primitive hierarchies can be checked if they are defined correctly by reviewing the is_part_of/has_part relationships between them, as in Figure 5.15. For example, "value set 12" is used in composite subcategory 1 rather than "value set 1" which is used in the supercategory for EVA1. "Value set 12" will have one of the primitive subcategories in Figure 5.13a as its range category, while "value set 1" has the primitive supercategory in Figure 5.13a as its range category.

5.6 Integration of Product and Process Models

Both product and process models are needed to design a structural system or component. Thus, the product and process models, that are developed in the previous section, should be integrated. As discussed earlier, the integration of product and process models means that each O-type entity describes information created and used by the corresponding A-type entity (i.e., design activity), and each A-type entity is described as operations on the corresponding O-type entity (i.e., design information). Both product and process models are developed using the consistent concepts and notation, thus integration of the two models can be achieved by including *data_interdependent* relationships between the O-type and A-type entity categories (Figure 5.16). These relationships are included by adding *data attributes* to the A-type entities (Figure 5.17). A data attribute is an OEVA in an A-type entity in the process model which identifies an associated O-type attribute in the

product model. This attribute is defined for a set of O-type and A-type entities that have a data_interdependent relationship. That is, a data attribute is added to a process model to allow a product model to represent data interdependence.

Note that, as shown in Figure 5.18, an O-type entity category for local data can also be included in a process model to allow a process model to operate on temporary design information not defined in the product model but needed for the process model. An O-type entity category for local data simplifies the interaction between the two models (i.e., product and process models). The O-type entity category for local data consists of three types of design information: (1) incoming parameters whose values come from attributes of O-type entities in the product model and are used by actions of A-type entities in the process model; (2) outgoing parameters whose values are generated by actions of A-type entities in the process model and are stored as attributes of O-type entities in the product model; and (3) local parameters for temporary data that is not associated with the O-type entities of a product model but is needed for actions in the process model. For incoming and outgoing parameters, a process model includes two important actions: (1) a retrieve action for incoming parameters and (2) an update action for outgoing parameters. That is, values for incoming parameters are available to the process model through the retrieve action and those for outgoing parameters are sent to the product model through the update action. Once values for incoming parameters are contained in the O-type entity for local data and then they are used by all associated actions in a process model. When values for outgoing parameters are determined through iteration of certain actions, these values are not sent to the product model after each iteration. The update action in the process model sends the final values to the product model. The values that are determined in each iteration are contained in the local data O-type entity.

The following four steps are used to integrate the product and process models.

1. Check the compatibility between composite categories from the two models, as in Figure 5.20. Note that a set of O-type instances from an O-type composite category are related to a set of A-type instances from a corresponding A-type composite category. That is, the correspondence between product and process models is between composite categories. Therefore, the compatibility is checked between the O-type and A-type composite categories. If the product and process models are not compatible, steps 1 through 4 in Section 5.5 should be repeated.

2. Establish the data_interdependent relationship between the compatible pairs of O-type and A-

type composite categories, as shown in Figure 5.16. Note that there are different ways to define data_interdependent relationships between product and process models. For example, data_interdependent relationships may be defined between primitive categories from the two models, or between composite categories. In this discussion, the data_interdependent relationship is established between composite categories from the two models to simplify the interaction between the two models. That is, design information of an O-type composite category is created and used by actions of the corresponding A-type composite category.

3. Formalize the data_interdependent relationship as a data attribute, and add this data attribute to the A-type composite category, as in Figure 5.17.

4. Create an O-type entity category for local data which has three types of attributes (i.e., incoming parameters; outgoing parameters; and local parameters) and add a corresponding OEVA to the A-type composite category, as in Figure 5.18. Figure 5.19 details Figure 5.18 showing how an O-type entity category for local data is used by the AEVAs that are parts of the higher level A-type entity category with the local data attribute. Note that entity categories from the range categories of the AEVAs that operate on the O-type entity category for local data also have the local data attribute, as shown in Figure 5.19.

5.7 Creation of Instance Models

The generalization models developed can be tested by creating instance models for specific design problems. An instance model is generated using categories defined in generalization models, as in Figure 5.21. By definition, instances are generated from a class. The O-type and A-type composite categories have corresponding O-type and A-type classes respectively. The data_interdependent relationship between the composite categories also applies to the classes. If instance models cannot be generated, all earlier steps in the procedure should be repeated. Note that the O-type and A-type instances are not uniquely mapped to each other, as shown in Figure 5.21. Each O-type instance is a set of values for the attributes of an O-type class. Each A-type instance is one activity of the design process. However, an activity may be repeated. Several instances of one activity could operate on the same O-type instance (i.e., the same set of values for

the attributes of an O-type entity). In other words, more than one A-type instance can be mapped to a single O-type instance.

The procedures for developing product and process generalization and instance models are summarized in Figure 5.22.

6. Application of Modeling Approach

The purpose of this chapter is to illustrate the development of a product model and a process model for a specific structural design problem using the procedures discussed in Chapter 5. The preliminary design of a beam is the problem considered. The scope of the models is as follows: (1) the beam design alternatives for only one beam design problem are considered; (2) O-type entities that represent information about the beam design problem and proposed solutions, and A-type entities that represent the preliminary design tasks of the MSD model are considered; (3) the instance model for a beam that spans 24 ft between simple supports, carries a uniform dead and live load of 600 lb/ft, and is braced at intervals of 6 ft is considered; and (4) wide flange beams and other types of flexural members are proposed as alternative solutions to this beam design problem, however wide flange beams are emphasized. The chapter is organized as follows. First, Section 6.1 presents a product generalization model for beam design. Then, Section 6.2 presents a process generalization model for beam design. Section 6.3 discusses integration of the two different models. Finally, Section 6.4 presents the instance model for the given beam design problem.

6.1 Development of Product Model for Beam Design

Figure 6.1 outlines an O-type entity category BEAM that describes information about a beam design problem and several proposed solutions. For a single beam design problem, attributes (e.g., load, span_length, deflection_limit, etc.) that describe the design problem are single valued or multi valued; only one design problem is described. Attributes (e.g., internal_forces, weight, cost, etc.) that describe proposed solutions are multi valued with as many values as the number of possible solutions that are considered.

The BEAM entity category can be refined using beam_problem and beam_solution B-OEVAs, as shown in Figure 6.2. The use of these B-OEVAs allows us to show that structural design is viewed as a process which separates information about the design problem from information about proposed solutions. The beam_problem B-OEVA and corresponding range category BEAM PROBLEM group together attributes such as load, span_length, unbraced_length, deflection_limit, and so on; and the beam_solution B-OEVA and corresponding range category BEAM SOLUTION

group together attributes such as internal_forces, weight, cost, material_density, elastic_modulus, section_area, and moment _of_inertia and so on. Thus, a product model for beam design consists of two broad types of entities: (1) entities that describe the beam design problem and (2) entities that describe proposed solutions. Figure 6.3 outlines entity categories that are used to describe the beam design problem in more detail. A LOAD entity describes a trapezoidal loading for a line element. Entities that describe several proposed solutions (i.e., design alternatives) can be developed using the procedure outlined in Section 5.5 as follows:

6.1.1 Build Composite Hierarchy

An O-type composite generalization hierarchy with entity categories for various beam design alternatives is shown in Figure 6.4. Generalization abstraction is used to define the BEAM SOLUTION category as an O-type supercategory for the set of O-type entity categories defined for the various beam design alternatives. That is, the BEAM SOLUTION category is a supercategory that has attributes for the properties that are common to all beam design alternatives, such as internal_forces, weight, cost, material_density, elastic_modulus, section_area, and moment _of_inertia.

6.1.2 Refine Supercategory using EVAs

The O-type supercategory BEAM SOLUTION is refined using elaborated_beam_problem, material, geometry, and merit BS-OEVAs, as shown in Figure 6.5. ELABORATED BEAM PROBLEM, BEAM MATERIAL, BEAM GEOMETRY, and BEAM MERIT are range categories defined for the BS-OEVAs. The elaborated_beam_problem BS-OEVA and corresponding range category ELABORATED BEAM PROBLEM includes internal_forces and other attributes; the material BS-OEVA and corresponding range category BEAM MATERIAL group together attributes such as material_density and elastic_modulus; the geometry BS-OEVA and corresponding range category BEAM GEOMETRY group together attributes such as section_area and moment_of_inertia; and the merit BS-OEVA and corresponding range category BEAM MERIT group together attributes such as weight and cost. Since these attributes are defined as BS-OEVAs, the subcategories of the BEAM SOLUTION category should redefine value sets for these attributes (i.e., value set specialization should be used).

Figure 6.6a shows the process of introducing the OEVAs into BEAM SOLUTION as a decomposition

of the BEAM SOLUTION category into identified parts. Figure 6.6b shows another description of the BEAM SOLUTION category that has been expanded to include several derived attributes. The values of these attributes are determined from the attributes of entities from the categories, ELABORATED BEAM PROBLEM, BEAM MATERIAL, BEAM GEOMETRY, BEAM MERIT,and so on. Thus, in this version of BEAM SOLUTION, values for these attributes are available from a BEAM SOLUTION entity as they were in the original description (Figure 6.4), however these value are derived from entities in the lower level categories.

6.1.3 Build Primitive Hierarchy

O-type primitive generalization hierarchies are developed for the range categories of the OEVAs identified in the second step. The range categories of the BEAM SOLUTION category are the supercategories of the primitive hierarchies. Subcategories of the primitive hierarchies are defined through a specialization process that adds attributes; a value type specialization process that redefines value types for inherited attributes; or a value set specialization process that redefines value sets for inherited attributes.

As discussed earlier in Section 5.4.4, primitive hierarchies can be defined independently or dependently. The elaborated beam problem, the beam material and the beam geometry primitive hierarchies (Figures 6.7, 6.9 and 6.10) are independent while the beam merit primitive hierarchy (Figure 6.10) is dependent on information described by categories in the beam material and beam geometry primitive hierarchies. The independent primitives hierarchies are discussed first and then the dependent primitive hierarchy is discussed.

Primitive hierarchies that are independent have entity categories whose attributes are independent of other primitive hierarchies. Figure 6.7 shows the elaborated beam problem primitive hierarchy, which has subcategories defined by adding attributes needed to describe beam design problems in more detail for different beam design alternatives. The ELABORATED BEAM PROBLEM supercategory has no attributes. The ELABORATED SHAPE BEAM PROBLEM subcategory has specific attributes such as bending_moment_diagram, shear_force_diagram, maximum_bending_moment, and maximum_shear_force; and the ELABORATED TRUSS BEAM PROBLEM subcategory has specific attributes such as member_forces.

Figure 6.8 shows a different elaborated beam problem primitive hierarchy, which has

subcategories defined through a value set specialization. The ELABORATED BEAM PROBLEM supercategory has an internal_forces attribute with an "internal forces" value set. The ELABORATED SHAPE BEAM PROBLEM subcategory redefines the value set for internal_forces attribute to be "BMD&SFD"; and the ELABORATED TRUSS BEAM PROBLEM subcategory redefines it to be "bar forces". Note that the use of the internal_forces OEVA and value set specialization appears to minimize the differences between subcategories in the elaborated beam problem primitive hierarchy. Entity categories INTERNAL FORCES, BMD&SFD, and BAR FORCES are range categories for the value sets "internal forces", "BMD&SFD", and "bar forces", respectively. These range categories can be organized into a generalization/specialization hierarchy because of the similarity in the role of these entities in the model, however, they have no common attributes, as shown in Figure 6.8b. The use of OEVAs and value set specialization in the subcategories of a primitive hierarchy has the benefit of providing a clearer and more uniform description of the subcategories of the primitive hierarchy by hiding more specific design properties within EVAs. This uniformity is not necessary, but can be provided when appropriate.

Figure 6.9 shows the beam material primitive hierarchy, which has subcategories defined by adding attributes that describe material properties. The BEAM MATERIAL supercategory has attributes, such as material_density and elastic_modulus, which are common to any type of beam material. The STEEL subcategory has specific attributes such as steel_type, steel_density, elastic_modulus_of_steel, and yield_strength_of_steel, while the CONCRETE subcategory has specific attributes such as concrete_type, and compressive_strength_of_concrete. As discussed earlier, a supercategory attribute with the DS-DVA value type requires that the subcategories provide a method of determining attribute values. For example, for the material_density attribute of the BEAM MATERIAL supercategory, the STEEL subcategory provides a method for determining the density which simply returns the value of the B-DVA steel_density, while the CONCRETE subcategory provides a method which determines the density based on the B-DVA concrete_type.

The beam geometry primitive hierarchy is shown in Figure 6.10. It is worth noting that the attributes of the WIDE FLANGE BEAM GEOMETRY category include only one base attribute, shape_designation, and many derived attributes (e.g., depth, web_thickness, flange_width, etc.). The values of these derived attributes are obtained from a table lookup in the AISC database of wide flange shapes (AISC 1989). Most of the attributes of the PLATE GIRDER GEOMETRY category (Figure 6.10) are base attributes whose values are specified during the design process.

Primitive hierarchies that are dependent have entity categories whose attributes are dependent on other primitive hierarchies. The beam merit primitive hierarchy (Figure 6.11) is dependent. The categories in this hierarchy must use information about beam material and geometry to determine values of the weight and cost attributes. The required methods and the required information for calculating the weight and cost depend on type of design alternative. For example, the BEAM MERIT supercategory has the attribute weight which has the DS-DVA type. The WIDE FLANGE BEAM MERIT subcategory redefines the value type of the inherited weight attribute to DE-DVA and provides a specific derivation procedure for wide flange beams; and the PLATE GIRDER MERIT subcategory also redefines the value type of the weight attribute to DE-DVA and provides a specific derivation procedure for plate girders. The DE-DVA type indicates that the attribute value depends on information from other entities. To provide this information, categories in the beam merit primitive hierarchy include the beam OEVA. Entities in the range categories of this attribute can provide information about beam material and geometry needed to calculate the weight. For example, the BEAM SOLUTION category is the range category for the beam OEVA. The BEAM SOLUTION category includes derived attributes such as material_density and section_area, that are derived from the BEAM MATERIAL, BEAM GEOMETRY categories, as shown in Figure 6.6b. Value set specialization of the beam OEVA allows the specific beam material and geometry information to be obtained for different design alternatives, as shown in Figure 6.11. For example, the beam OEVA of the WIDE FLANGE BEAM MERIT category can provide the beam material and geometry information required to calculate the weight of wide flange beams. Thus, the WIDE FLANGE BEAM MERIT subcategory redefines the value set for the beam attribute to be "wide flange beam", while the PLATE GIRDER MERIT subcategory redefines the value set for the beam attribute to be "plate girder".

The BEAM MERIT category obtains information from the BEAM SOLUTION category through the is_part_of relationship between the BEAM MERIT category and the BEAM SOLUTION category, as shown in Figure 6.12. Note that the relationship between BEAM MERIT and BEAM SOLUTION is two way: (1) the has_part relationship from BEAM SOLUTION to BEAM MERIT (Figure 6.6) created when the BEAM SOLUTION category is decomposed into identified parts; and (2) the is_part_of relationship from BEAM MERIT to BEAM SOLUTION (Figure 6.12) is required for the BEAM MERIT to access beam material and geometry information from the BEAM SOLUTION category. In a similar way, the two way relationship (i.e., has_part/is_part_of) is defined between subcategories of the

beam merit primitive hierarchy and the corresponding subcategories of the composite hierarchy for beam design alternatives, as shown in Figure 6.13.

6.1.4 Complete Composite Hierarchy

The O-type composite generalization hierarchy for beam design alternatives is completed by adding appropriate value sets to the subcategories shown in Figure 6.5, as shown in Figure 6.14. Relationships between the primitive and composite generalization hierarchies are reviewed to check if all O-type primitive categories are defined or not, as in Figure 6.15. These primitive and composite hierarchies together comprise a product generalization model for beam design alternatives. Finally, a product generalization model for beam design can be completed by putting the entities for the beam design problem together with the hierarchies of entities for the beam design alternatives.

6.2 Development of Process Model for Beam Design

A process model for beam design that corresponds to the product model in the previous section can be developed using a similar procedure. Figure 6.16 outlines an A-type entity category BEAM DESIGN that describes design tasks for the preliminary design of a beam. For a single beam design problem, the formulate_problem design task occurs only once because only one design problem is formulated, while design tasks that develop the properties of proposed solutions (e.g., calculate_internal_forces, size_member, etc.) occur more than once because these tasks are done for each of several proposed solutions.

The BEAM DESIGN entity category can be refined using beam_selection and beam_development B-AEVAs, as shown in Figure 6.17. The use of these B-AEVAs allows us to show the design tasks for the design of a beam that are identified in the MSD model (Sause and Powell 1991). The beam_selection B-AEVA and corresponding range category BEAM SELECTION includes formulate_problem and other attributes involved in the selection activity of the MSD model; and the beam_development B-AEVA and corresponding range category BEAM DEVELOPMENT group together attributes such as calculate_internal_forces, assume_allowable_stresses, size_member, and others involved in the development activity of the MSD model. Therefore, a process model for

beam design should include two broad types of entities: (1) entities for design tasks involved in selection (Figure 6.18), and (2) entities for design tasks that develop proposed solutions (i.e., design alternatives). Entities for design tasks that develop design alternatives can be developed using the procedure outlined in Section 5.5 as follows.

6.2.1 Build Composite Hierarchy

An A-type composite generalization hierarchy with entity categories for the design of various beam design alternatives is shown in Figure 6.19. Generalization abstraction is used to define the BEAM DEVELOPMENT category as an A-type supercategory for the set of A-type entity categories defined for the design of the various beam alternatives. That is, the BEAM DEVELOPMENT category is a supercategory that has attributes for the design tasks that are common to the design of all beam design alternatives, such as calculate_internal_forces, assume_allowable_stresses, size_member, review_member_for_internal_forces, and review_member_for_deflection.

6.2.2 Refine Supercategory using EVAs

The A-type supercategory BEAM DEVELOPMENT is refined using elaboration, design_proposal, and review BS-AEVAs, as shown in Figure 6.20. These BS-AEVAs are based on design tasks that are identified in the MSD model for preliminary development of a design alternative (Sause and Powell 1990). BEAM ELABORATION, BEAM DESIGN PROPOSAL, and BEAM REVIEW are range categories defined for the BS-AEVAs. The elaboration BS-AEVA and corresponding range category BEAM ELABORATION group together attributes such as calculate_internal_forces and assume_allowable_stresses; the design_proposal BS-AEVA and corresponding range category BEAM DESIGN PROPOSAL include the attribute size_member; and the review BS-AEVA and corresponding range category BEAM REVIEW group together attributes such as review_member_for_internal_forces and review_member_for_deflection. Figure 6.21 shows the grouping process as a decomposition of the BEAM DEVELOPMENT supercategory into identified parts.

6.2.3 Build Primitive Hierarchy

A-type primitive generalization hierarchies are developed for the range categories of the AEVAs identified in the second step. The range categories of the BEAM DEVELOPMENT supercategory are

supercategories of the primitive hierarchies. Subcategories of the primitive hierarchies are defined through a specialization process that adds more specific attributes (i.e., design tasks); a value type specialization process that redefines value types for inherited attributes; or a value set specialization process that redefines value sets for inherited attributes.

Figure 6.22a shows the beam elaboration primitive hierarchy. The BEAM ELABORATION supercategory has attributes, such as calculate_internal_forces, and assume_allowable_stresses, which are common to any type of beam elaboration. However, these attributes (i.e., design tasks) are not detailed enough and lower level, detailed design tasks are provided, which are different for the different subcategories of the BEAM ELABORATION category. Value set specialization is used to define subcategories of the beam elaboration primitive hierarchy. The BEAM ELABORATION supercategory has the value sets "internal force calculation" and "allowable stress calculation" for its attributes. The SHAPE BEAM ELABORATION subcategory redefines them to be "M&V calculation" and "Fb'&Fv' calculation"; and the TRUSS BEAM ELABORATION subcategory redefines them to be "axial force calculation" and "Fc'&Ft' calculation".

Figure 6.22b shows a generalization/specialization tree for the range categories defined for the calculate_internal_forces AEVA, and Figure 6.22c shows a generalization/specialization tree for he range categories defined for the assume_allowable_stresses AEVA. The M&V CALCULATION category in Figure 6.22b represents design tasks for calculating internal forces using AEVAs. Range categories for these AEVAs are shown in Figure 6.22d.

Figure 6.23 shows a different beam elaboration primitive hierarchy, which does not use the additional generalization/specialization trees shown in Figure 6.22b and Figure 6.22c. This primitive hierarchy is not as uniform as the one in Figure 6.22, and the supercategory has no attributes. However, it is simpler than the one in Figure 6.22, and since each category in the hierarchy describes the same type of primitive concept, it is an acceptable primitive hierarchy.

Figure 6.24a shows the beam design proposal primitive hierarchy which consists of subcategories obtained by value set specialization. The BEAM DESIGN PROPOSAL supercategory has the size_member attribute (which is common to any type of beam design proposal) with the value set "size member". The WIDE FLANGE BEAM DESIGN PROPOSAL subcategory redefines the value set to be "wide flange beam size member"; and the PLATE GIRDER DESIGN PROPOSAL subcategory redefines the value set to be "plate girder size member". Figure 6.24b shows a

generalization/specialization tree for the range categories defined for the size_member AEVA. The PLATE GIRDER SIZE MEMBER in Figure 6.24b describes design tasks for sizing a plate girder by using AEVAs such as propose_web_size and propose_flange_size, and the range categories for these AEVAs are shown in Figure 6.24c. Figure 6.25 shows a different beam design proposal primitive hierarchy, which does not use the additional generalization/specialization trees shown in Figure 6.24b and 6.24c.

Figure 6.26a shows the beam review primitive hierarchy which consists of subcategories obtained by value set specialization. The BEAM REVIEW supercategory has common attributes such as review_member_for_internal_forces, review_member_for_deflection, and so on. The review_member_for_internal_forces attribute is an AEVA. The BEAM REVIEW supercategory has the value set "review member for internal forces" for the AEVA review_member_internal_forces. The DOUBLY SYMMETRIC ROLLED BEAM REVIEW subcategory redefines it to be "bend & shear review for doubly symmetric rolled beams"; the SINGLY SYMMETRIC ROLLED BEAM REVIEW subcategory redefines it to be "bend & shear review for singly symmetric rolled beams"; the BUILT-UP BEAM REVIEW subcategory redefines it to be "bend & shear review for built-up beams"; and the TRUSS BEAM REVIEW subcategory redefines it to be "compression & tension review". Figure 6.26b shows a generalization/specialization tree for the range categories defined for the review_member_for_internal_forces AEVA. Since review is similar for the internal forces of members such as doubly or singly symmetric rolled beams or built-up beams, an entity category REVIEW MEMBER FOR INTERNAL FORCES is included for these common design tasks, as shown in Figure 6.26b. Figure 6.27 shows a different beam review primitive hierarchy that does not use the additional generalization/specialization trees shown in Figure 6.26b.

As discussed earlier in Chapter 4, value sets for the AVAs of a process model should be represented in terms of action specifications and these action specifications must be included in a process model. The action specifications for wide flange beam design will be discussed in Section 6.4.

6.2.4 Complete Composite Hierarchy

The A-type composite generalization hierarchy for beam development is completed by adding appropriate value sets to the subcategories shown in Figure 6.20, as shown in Figure 6.28. Relationships between the primitive and composite generalization hierarchies are reviewed to check

if all A-type primitive entity categories are defined or not, as in Figure 6.29. These primitive and composite hierarchies together comprise a process generalization model for the development of beam design alternatives. Finally, a process generalization model for beam design can be completed by putting the entities for the design tasks involved in beam selection together with the hierarchies of entities for the design tasks involved in the development of the beam design alternatives, as shown in Figure 6.30b.

6.3 Integration of Product and Process Models for Beam Design

The product and process models for beam design, that are partially developed in the previous sections, are integrated in this section. The integration of the product and process models for beam design can be achieved using the procedure outlined in Section 5.6 as follows:

1. The compatibility between composite categories from the product and process generalization models is checked, as shown in Figure 6.30. For each A-type composite category in the process model, there should be a corresponding O-type category in the product model and vice versa. For example, the O-type composite category WIDE FLANGE BEAM corresponds to the A-type composite category WIDE FLANGE BEAM DEVELOPMENT. If the compatibility cannot be established, steps discussed in Sections 6.1 and 6.2 should be repeated.

2. The relationship data_interdependent is established between the O-type and A-type composite entities. The O-type entities that describe the beam problem and the A-type entities that describe beam selection design tasks, as well as the O-type entities that describe beam solutions (design alternatives) and the A-type entities that describe the development tasks for beam alternatives all have data_interdependent relationships. The data_interdependent relationships are defined between the BEAM PROBLEM and the BEAM SELECTION categories; between the BEAM SOLUTION and the BEAM SELECTION categories; between the BEAM PROBLEM and the BEAM DEVELOPMENT categories; between the BEAM SOLUTION and the BEAM DEVELOPMENT categories; and between the WIDE FLANGE BEAM and the WIDE FLANGE BEAM DEVELOPMENT categories, as shown in Figure 6.31. If other beam alternatives were shown in Figure 6.31, the data_dependent relationship would be defined between the corresponding beam solution and beam development categories.

3. The data_interdependent relationships are formalized as data attributes beam_problem

and beam_solution, and these attributes are added to the corresponding A-type entities, as shown in Figure 6.32.

4. A local data OEVA for wide flange beam design is added to the WIDE FLANGE BEAM DEVELOPMENT category, and the corresponding range category WF BEAM LOCAL DATA is added to the model.

6.4 Creation of An Instance Model for Specific Beam Design Problem

The generalization models developed in the previous sections can be tested by creating instance models for specific beam design problems. An instance model is generated using categories defined in the generalization models. Since instances are generated from a class, the classes that correspond to the categories for wide flange beam design are discussed first. Figures 6.33a and 6.33b show O-type and A-type classes, respectively, that are needed for wide flange beam design. Note that the O-type class WF BEAM LOCAL DATA is also included in the process model. Figure 6.33c shows the WF BEAM LOCAL DATA class with attributes for incoming parameters, outgoing parameters, and local parameters. Values of incoming parameters are retrieved from the instances of the O-type classes in the product model by the retrieve action of an instance of the WF BEAM DEVELOPMENT class. Values of the outgoing parameters are generated by actions of instances of the A-type classes and stored in an instance of the WF BEAM LOCAL DATA class and these values are sent to the product model by the update action of the WF BEAM DEVELOPMENT class; and values of local parameters are passed between the instance of the WF BEAM LOCAL DATA class and the actions of the A-type instances.

Figure 6.34 provides a detailed description of O-type and A-type classes related to beam selection tasks; Figure 6.35 provides details of O-type and A-type classes related to elaboration of the design problem for wide flange beams; Figure 6.36 provides details of O-type and A-type classes related to design proposal for wide flange beams; and Figure 6.37 provides details of O-type and A-type classes related to design review for wide flange beams.

Instances for wide flange beam design are created from the O-type and A-type classes shown in Figures 6.34 through 6.36. Figure 6.38 shows a set of instances for wide flange beam design. Figures 6.39 through 6.42 show how values of O-type instances are used and generated by A-type

instances in more detail. Actions for A-type instances include specific expressions and values.

Values of the attributes of an O-type WIDE FLANGE BEAM instance (e.g., WF beam1) are determined by actions of an A-type WIDE FLANGE BEAM DEVELOPMENT instance using the M_R method (AISC 1989) shown in Figures 6.35 through 6.37. Each O-type WIDE FLANGE BEAM instance uses a set of values for the attributes defined for the O-type WIDE FLANGE BEAM class to describe a wide flange beam design alternative. Each A-type WIDE FLANGE BEAM DEVELOPMENT instance provides actions for one pass through the wide flange beam development process.

Note that the values for outgoing parameters, which are determined through iteration of certain activities of the A-type entities, are not sent to O-type entities in the product model after each iteration. The values that are determined in each iteration are held in the WF BEAM LOCAL DATA entity. The update activity sends the values to the O-type entities in the product model. For example, a value for the attribute shape_designation of the O-type WF beam geometry1 instance is determined through iteration of the actions of multiple instances of the WF BEAM DESIGN PROPOSAL class and the DOUBLY SYMMETRIC ROLLED BEAM REVIEW classes. The value for this attribute is stored in the attribute W of the WF beam local data1 instance, not in the WF beam geometry1 instance. This is shown in Figure 6.41 where the WF beam geometry1 instance does not have the value for the shape_designation attribute while the WF beam local data1 instance has the value W10*26 for the W attribute. This value is reviewed by the actions of the doubly symmetric rolled beam review1 instance, and the design conditions are satisfied, the shape designation value is kept in the attribute W of the WF beam local data1 instance and sent to the shape_designation attribute of the WF beam geometry1 instance by an update action. This is shown in Figure 6.42 where the WF beam geometry1 instance has the value for the shape_designation attribute and the WF beam local data1 instance also has the value for the W attribute.

7. Summary and Conclusions

The report has investigated the development of formal and consistent modeling concepts for integrated structural design product and process models. These models are needed to describe and organize design information and design activities as part of the development of integrated computer-aided design systems. The report focused on a formal and consistent approach based on the concepts of semantic data models. The purpose of this chapter is to summarize the contents of the report and discuss further research that is required to develop complete product and process models.

7.1 Summary

The research approach adopted in this study included four main tasks: (1) study of information modeling concepts, in particular the concepts of semantic models; (2) development of elements needed to create formal product and process models; (3) development of an approach to generate product and process models for structural design; and (4) application of product and process modeling approach to a specific structural design problem. The results of these four steps were presented in Chapters 3 through 6. Chapter 2 briefly reviewed previous product and process model research.

The first step in the study was to review information modeling concepts from the field of computer science. These concepts proved to be useful for organizing design information and design activities in the field of structural engineering. Semantic data model concepts (i.e., entities, relationships, and attributes) and related abstraction mechanisms were the primary information modeling concepts reviewed in Chapter 3.

The second step in this study was to define elements needed to develop product and process models for structural design. This work also involved the study of graphical conventions for representing the models. Chapter 4 defined useful elements for product and process models that include entities, relationships, attributes, classes, instances, supercategories, subcategories, compound categories, and simple categories; and also discussed graphical representations

including the modified E-R diagram, the table representation, and the modified SDM diagram.

The third step in this study was to develop an approach for generating product and process models for structural design using the elements developed in the previous step. This work involved (1) understanding the characteristics of design alternatives considered in structural design problems; (2) identifying requirements for product and process models for structural design; (3) developing an approach and strategy for developing product and process models; and (4) developing procedures for generating product and process models and for integrating the two models. The results of this effort were presented in Chapter 5.

The final step in this study was to apply the modeling approach to generate specific product and process models. Chapter 6 presented product and process models for preliminary design of beams.

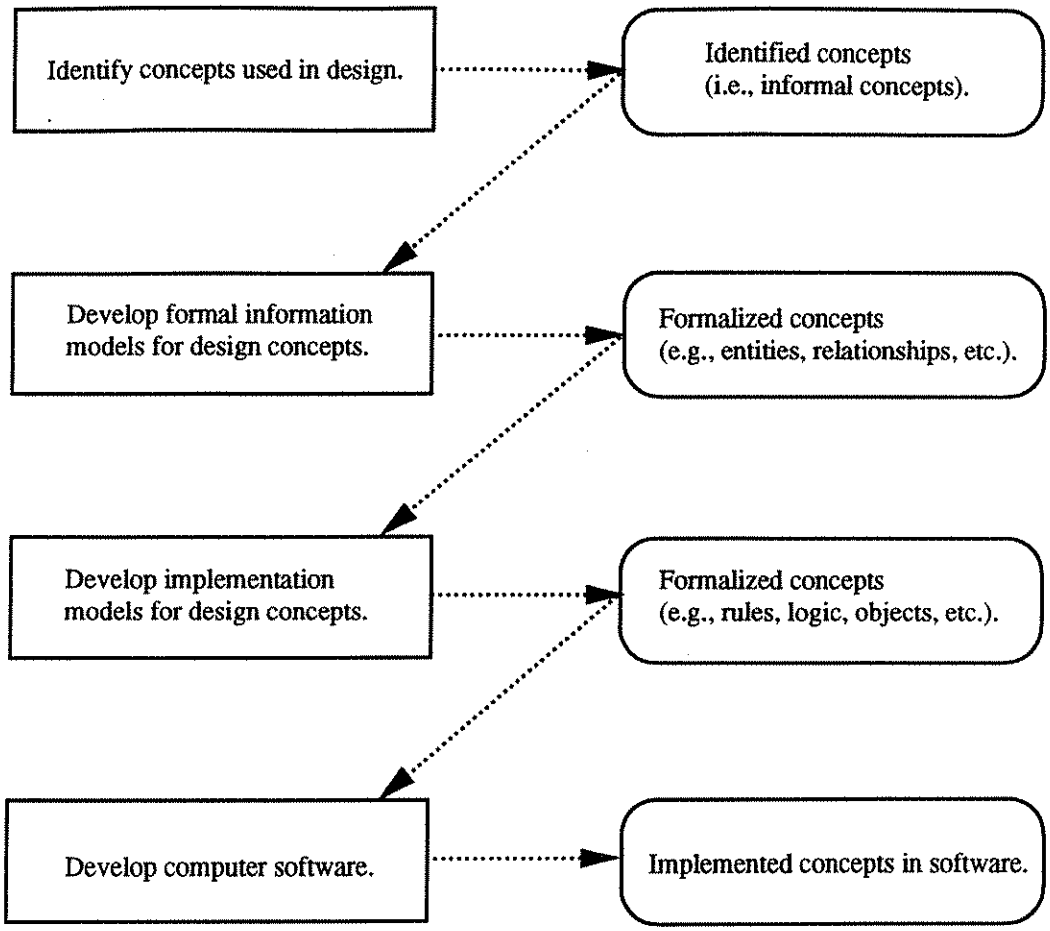
7.2 Summary of Recommendations for Further Research

This report has presented concepts, notation, and an approach to developing integrated product and process models for structural design. The specific models developed in the report are limited to models that can represent the preliminary development of several design alternatives for a beam design problem. However, the concepts and notation developed in this report appear to be sufficiently general to be used in the development of design alternatives for complete structural systems. Further research is needed to extend the concepts presented in the report to: (1) the design of complex structural systems; and (2) more comprehensive models that include (if possible) all entities and activities needed to design complete structural systems.

7.3 Concluding Remarks

A key feature of this work is the use of consistent concepts and notation for both product and process models. This consistent approach has several benefits: (1) product and process models can be developed together using similar concepts; (2) product and process models developed individually can be more easily integrated; and (3) product and process models can be implemented using the the same programing concepts and languages.

There are many steps involved in developing integrated computer-aided design systems. The work toward product and process models for structural design presented in this report is a useful step toward integrated computer-aided design systems. The value of this type of research is that it provides the developers of these systems with formal tools for describing and organizing structural design information and activities as the foundation for new systems.



a. Steps .

b. Information.

note : the dashed lines show relationships between steps and information and the arrows indicate that information is either generated or used

Figure 1.1. Steps toward the development of computer-aided design systems.

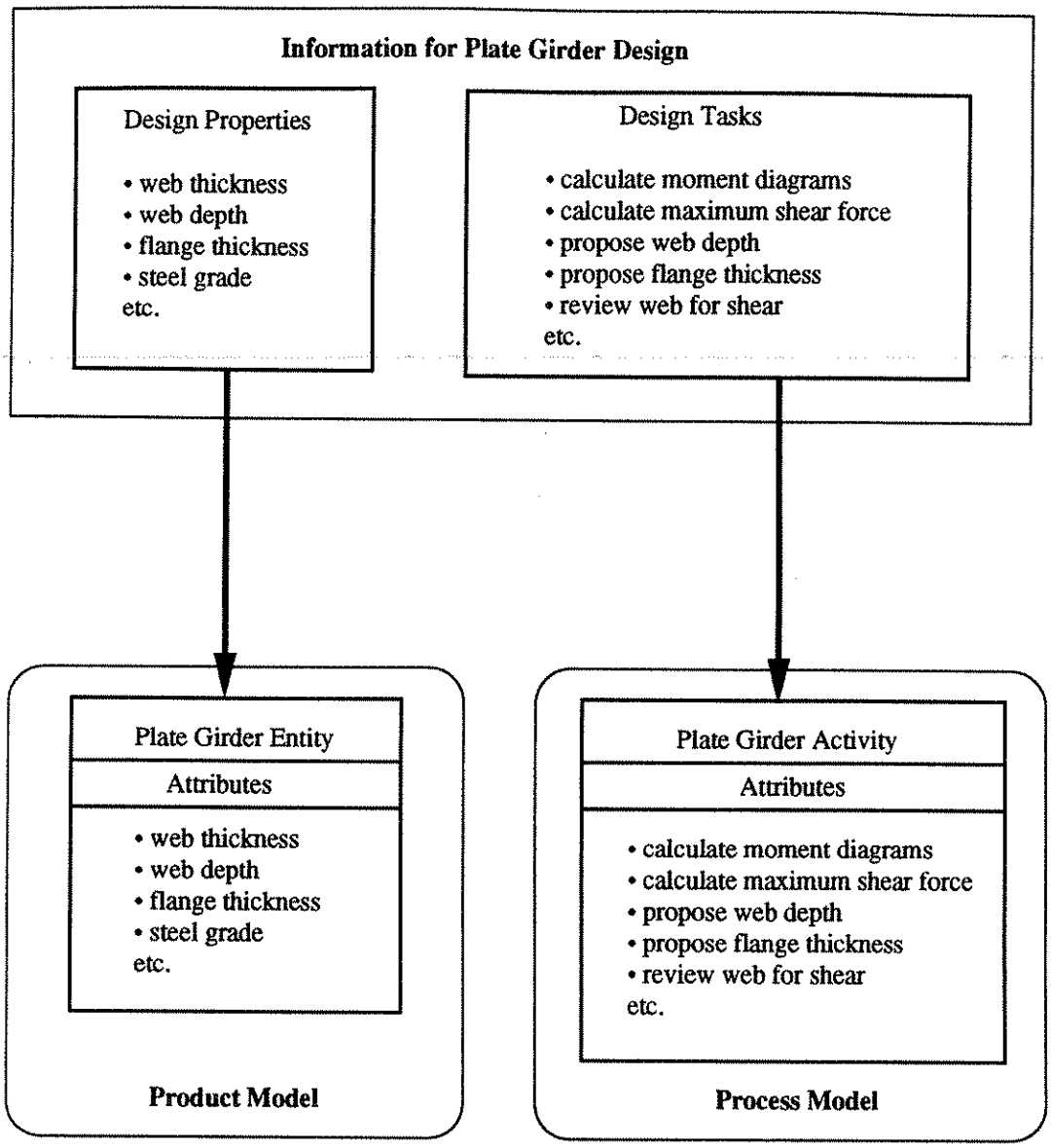


Figure 2.1. Mapping plate girder design information to product model and process model.

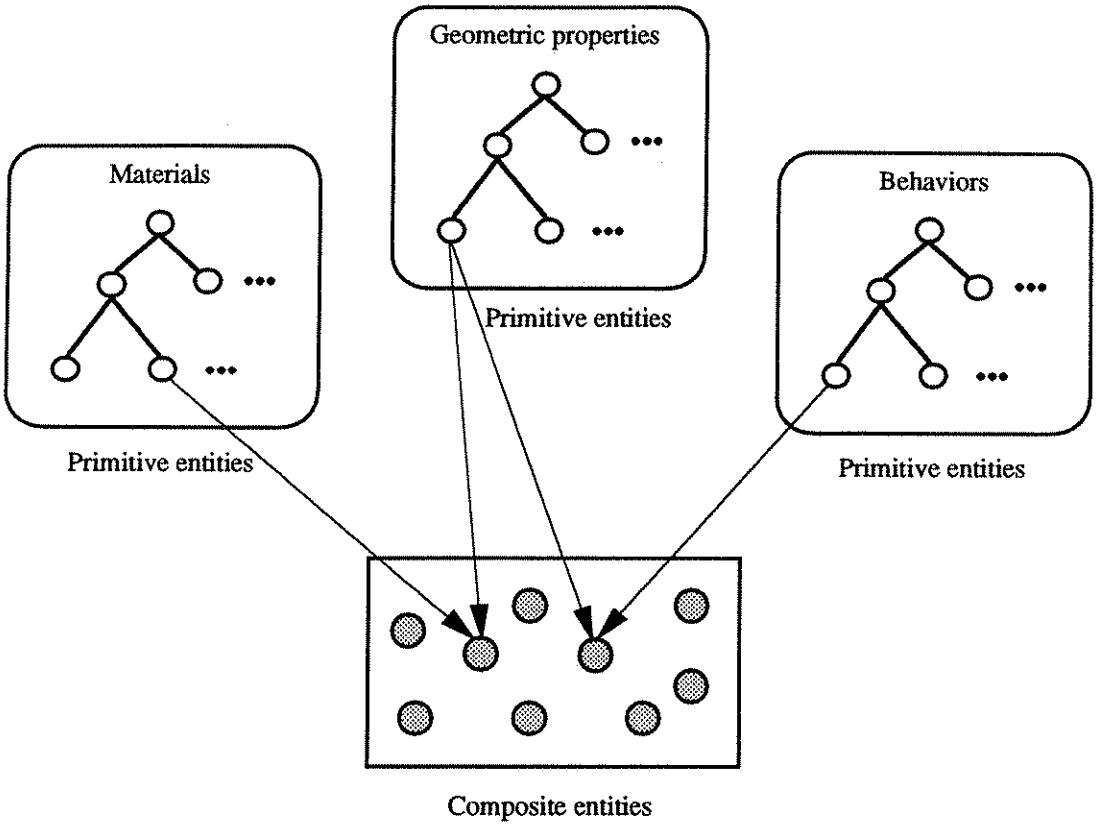
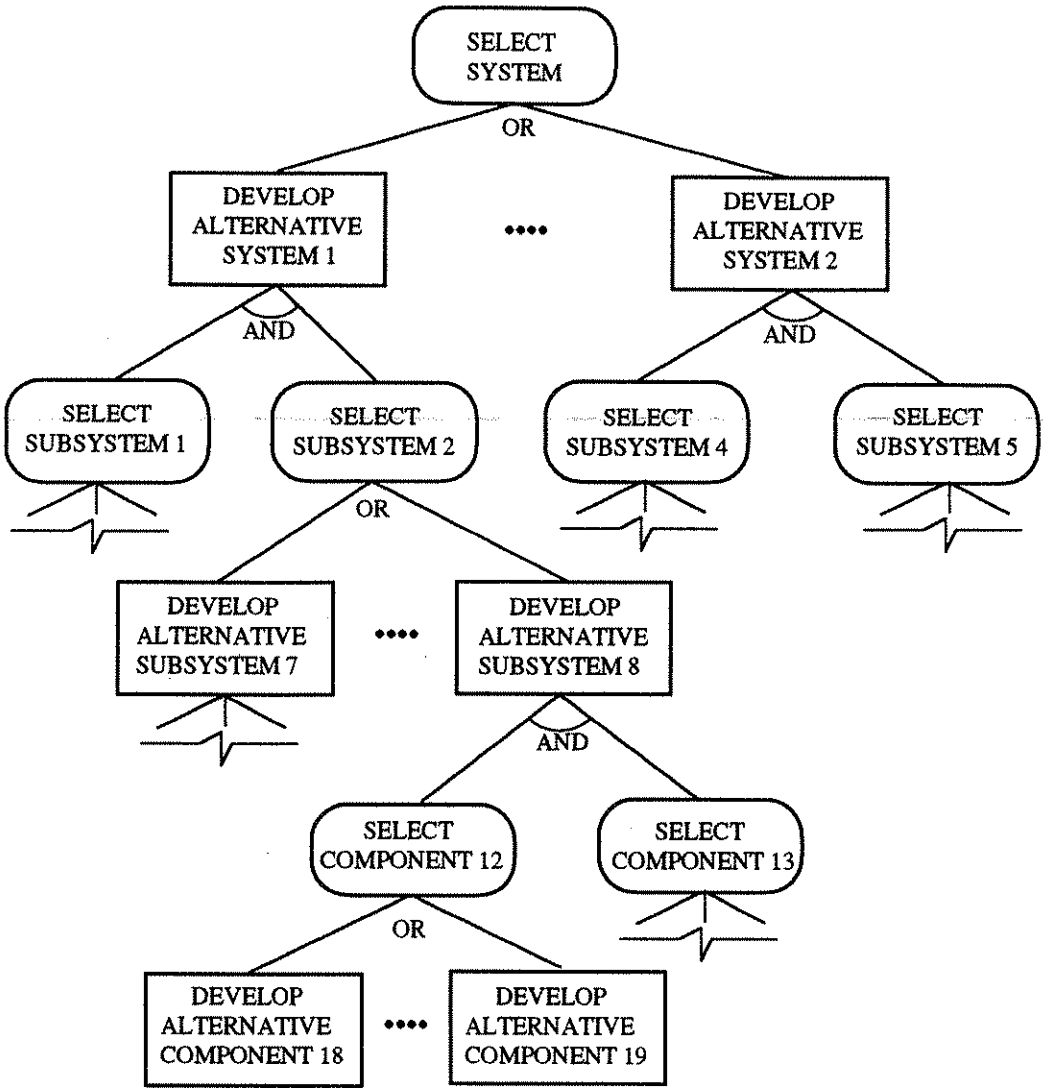


Figure 2.2. Composite entities aggregated from primitive entities.



Design Stage	**SELECTION** Tasks
Concept. I	• Formulation • Alternative Identification
Concept. II	• Formulation • Ranking • Elimination
Preliminary	• Formulation • Ranking • Elimination
Detailed	• Formulation • Selection • Completion

Design Stage	**DEVELOPMENT** Tasks
Concept. I	• Elaboration • Heuristic Evaluation
Preliminary I	• Elaboration • Design Proposal • Review
Preliminary II	• Elaboration • Component Problem Introduction • Analysis • Component Grouping • Component Design • Aggregation
Detailed	• Elaboration • Analysis • Component Design • Aggregation

Figure 2.3. Summary of MSD model.

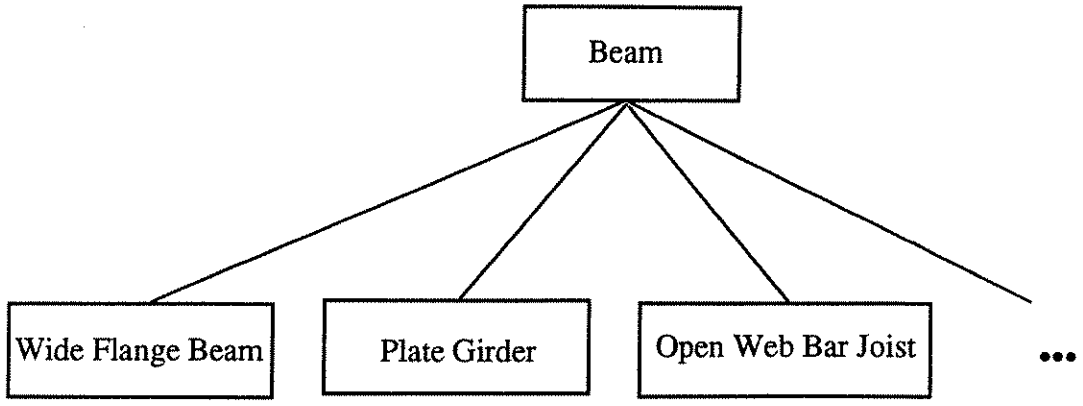


Figure 2.4. Beam product generalization model.

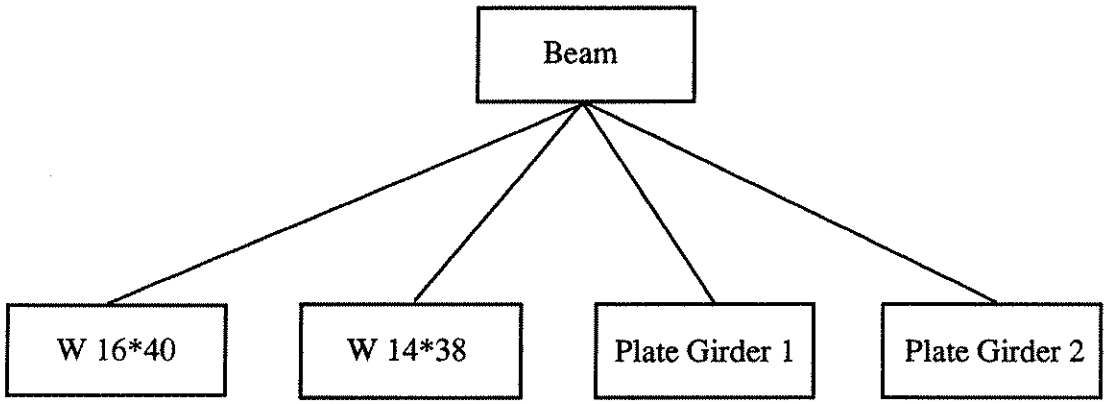


Figure 2.5. Beam product instance model.

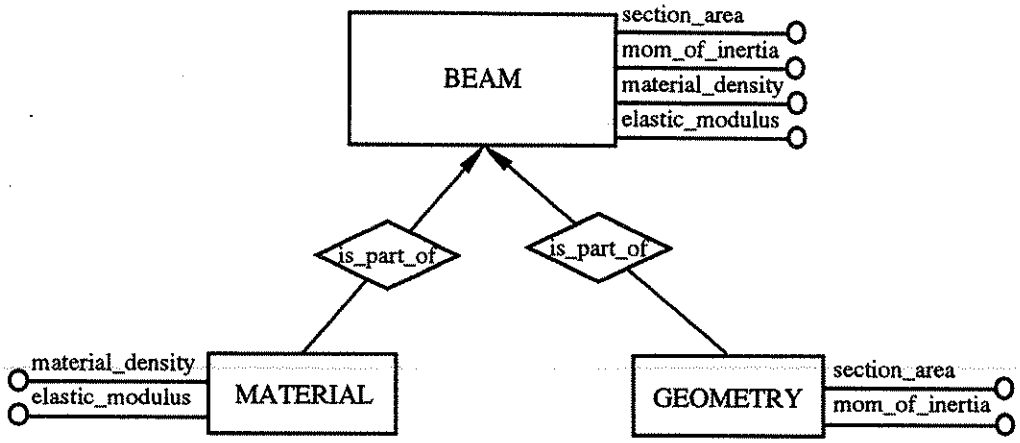


Figure 3.1. An example of the E-R model.

MATERIAL	beam_name	material_density	elastic_modulus

a. Relational table for beam material.

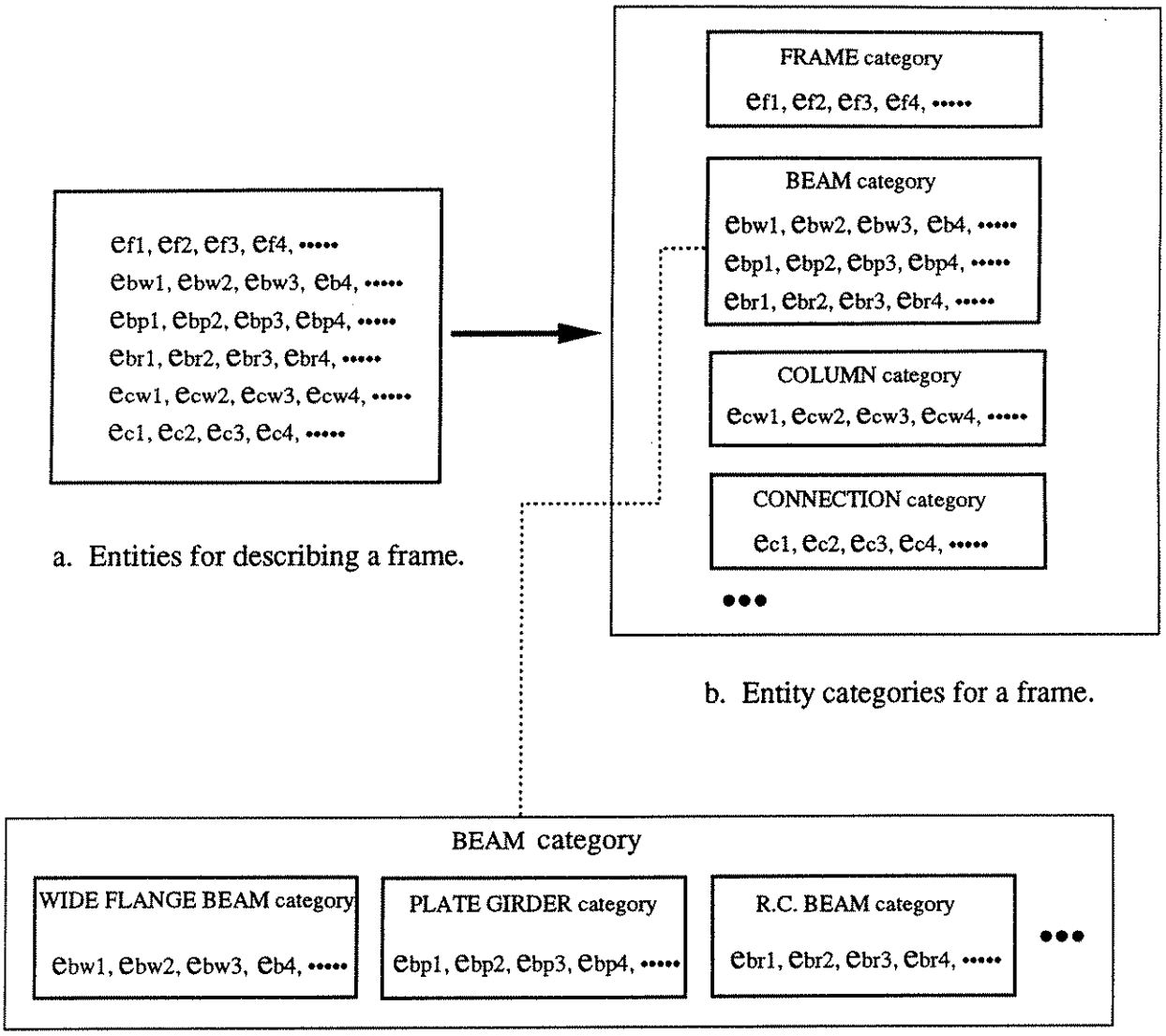
GEOMTRY	beam_name	section_area	mom_of_inertia

b. Relational table for beam geometry.

BEAM	beam_name	matl_density	elastic_mod	sect_area	mom_of_inertia

c. Relational table for beam .

Figure 3.2. An example of the relational model.

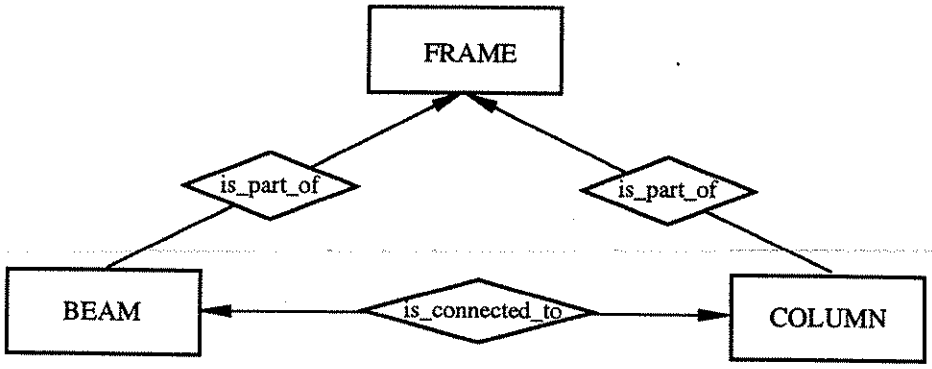


a. Entities for describing a frame.

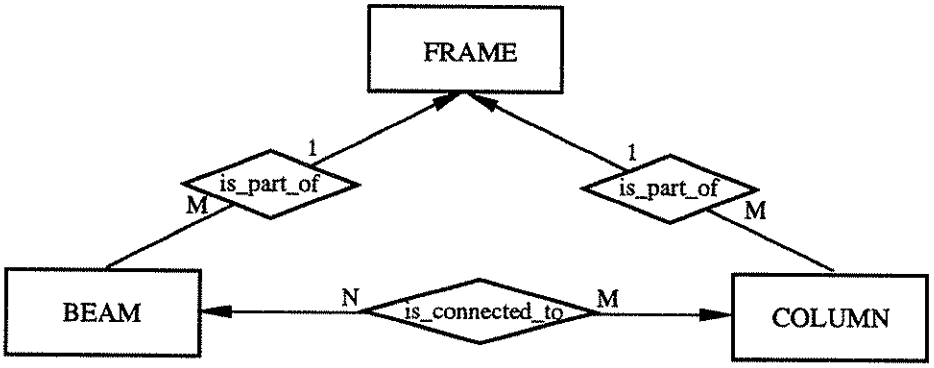
b. Entity categories for a frame.

c. Lower level entity categories of the beam category.

Figure 3.3. Entities and entity categories for describing a frame.

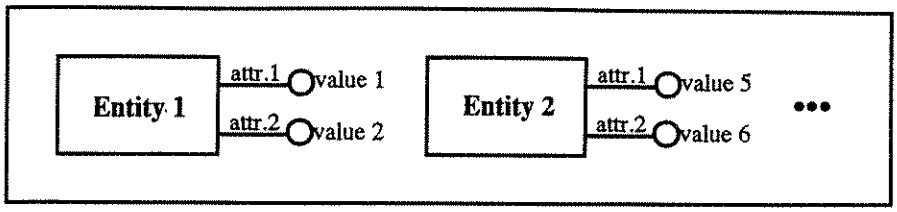


a. Frame representation in terms of entities and relationships.

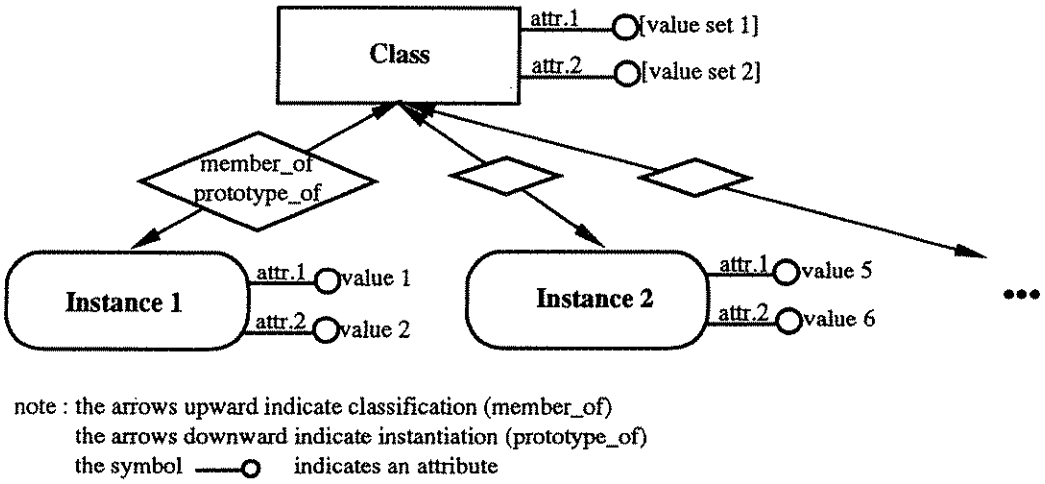


b. Frame representation showing cardinalities for relationships.

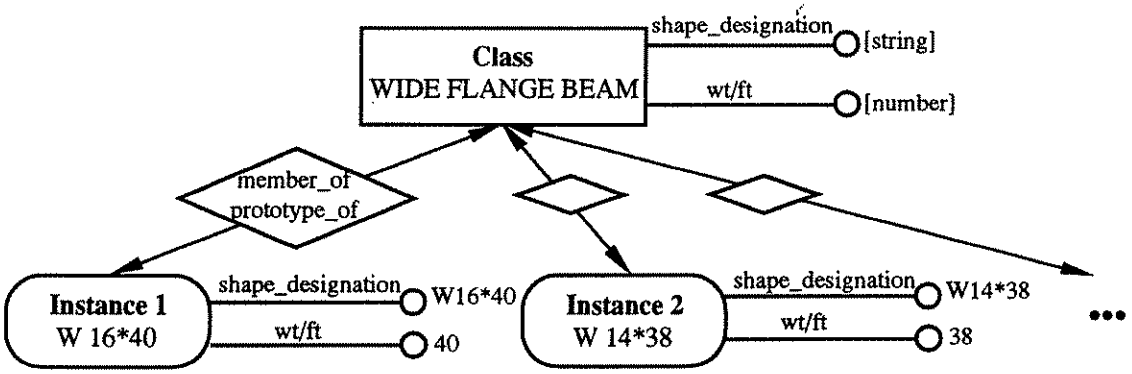
Figure 3.4. Frame representation.



a. A set of entities that have the same attributes.

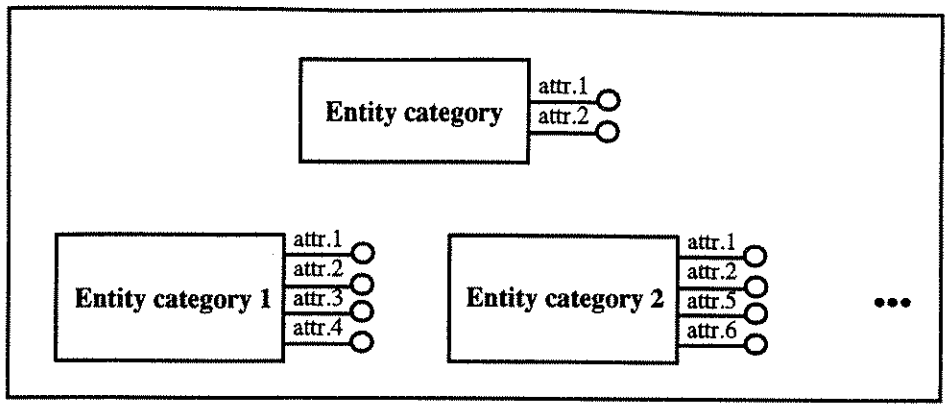


b. A tree for classification/instantiation.

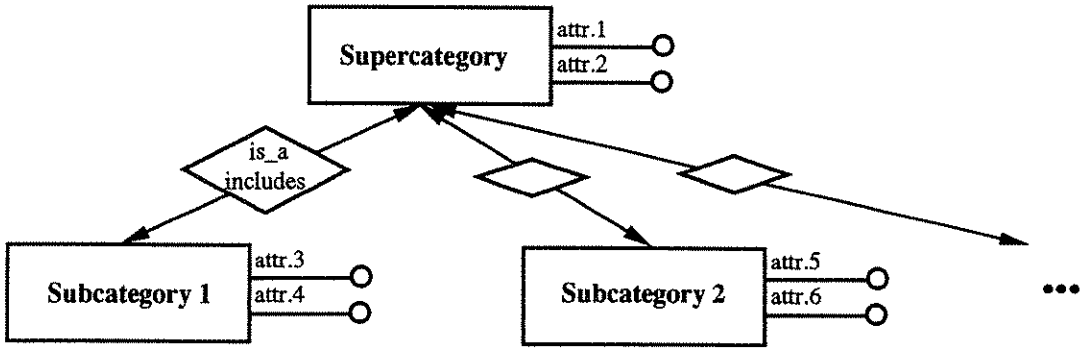


c. An example of classification/instantiation.

Figure 3.5. Classification/instantiation process.

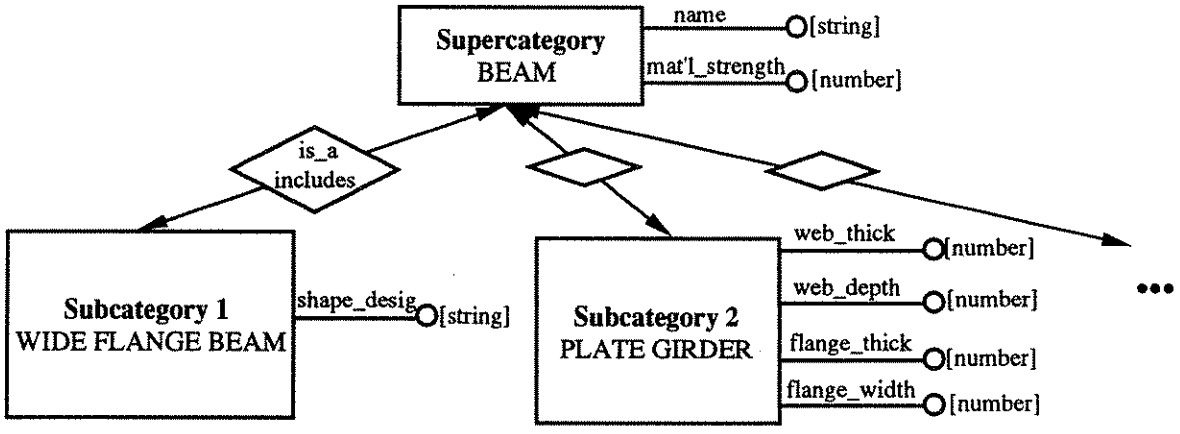


a. A set of entity categories that have common attributes.



note : the arrows upward indicate generalization (is_a)
the arrows downward indicate specialization (includes)

b. A tree for generalization/specialization.



c. An example of generalization/specialization.

Figure 3.6. Generalization/specialization process.

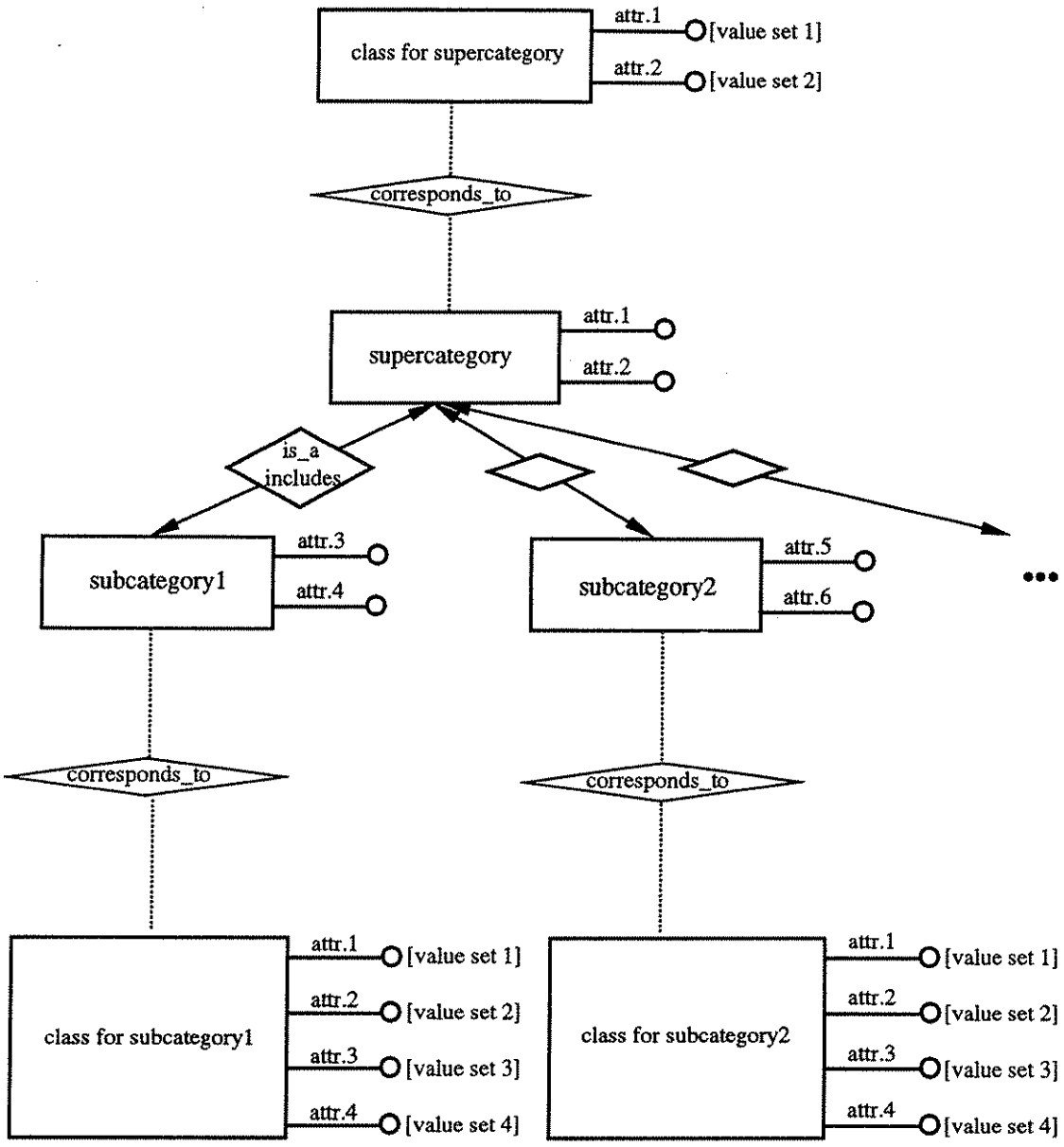
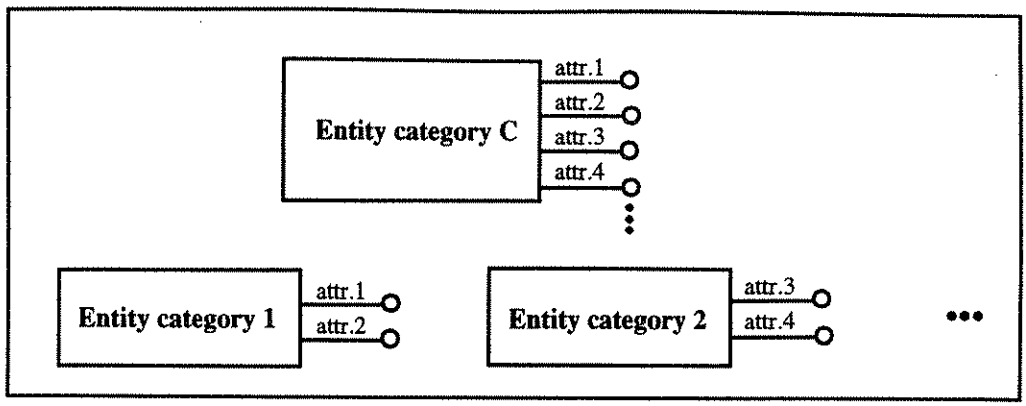
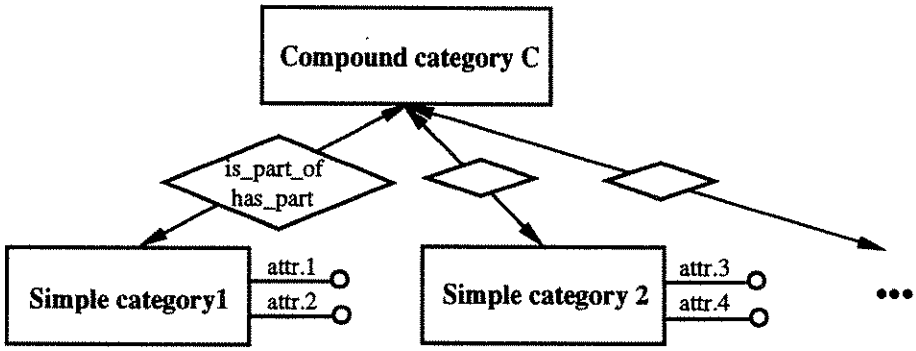


Figure 3.7. Classes for a supercategory and subcategories.

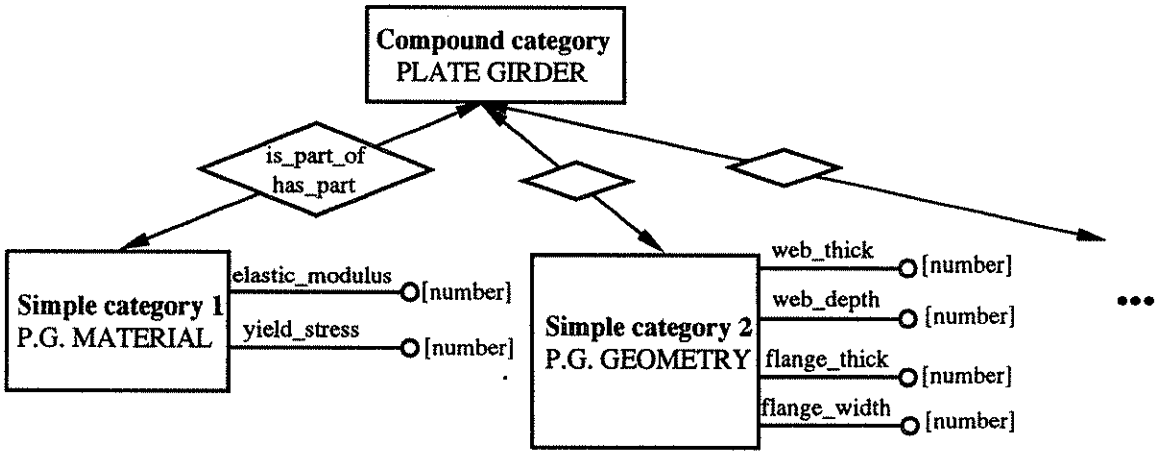


a.



note : the arrows upward indicate aggregation (is_part_of)
the arrows downward indicate decomposition (has_part)

b. A tree for aggregation/decomposition.



c. An example of aggregation/decomposition.

Figure 3.8. Aggregation/decomposition process.

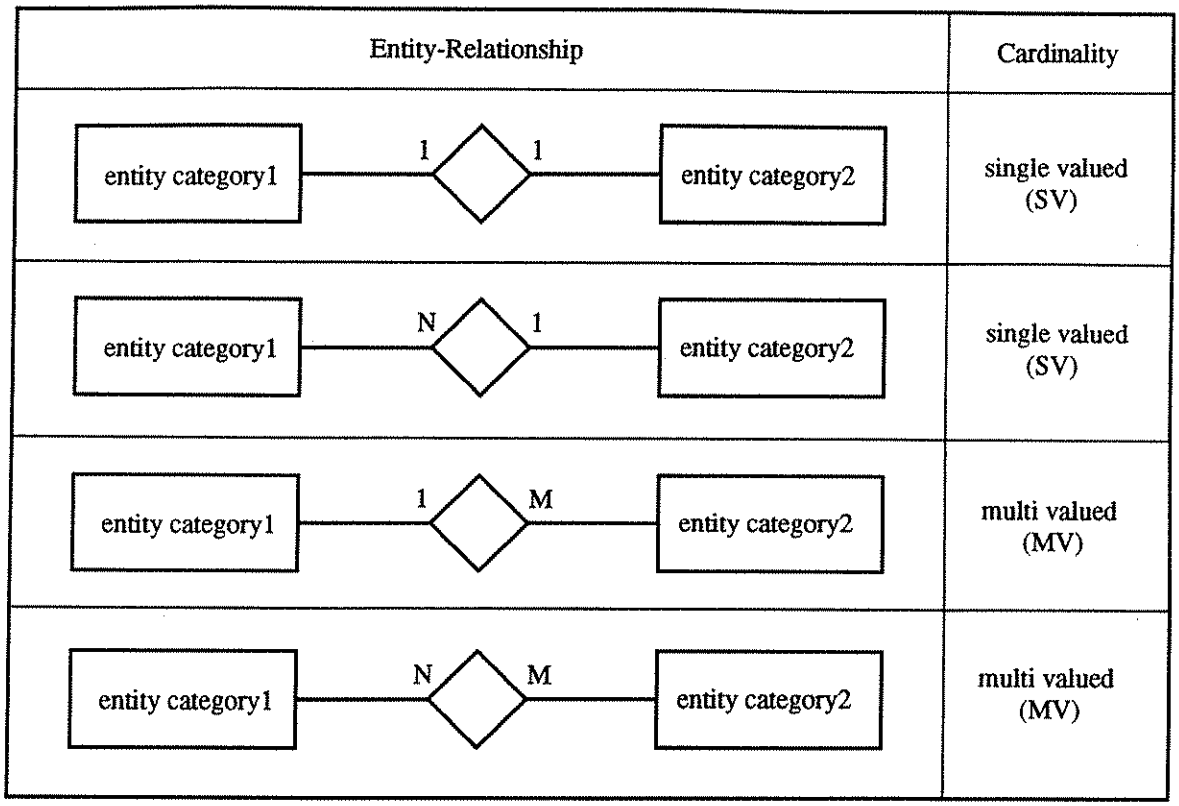
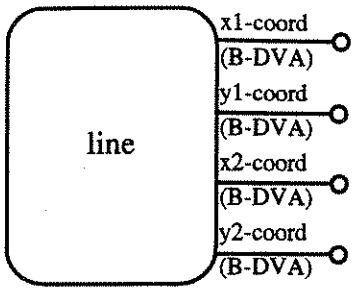
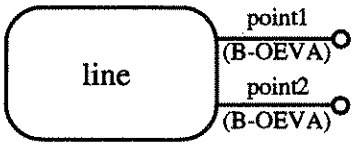


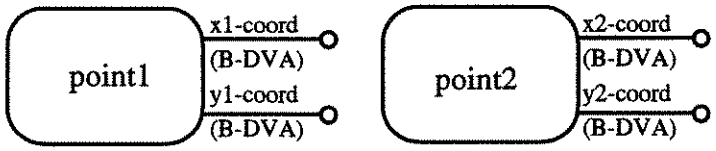
Figure 4.1. Cardinality table for relationships between two entity categories.



a. Line entity with data-valued attributes.



b. Line entity with entity-valued attributes.



c. Point entities with data-valued attributes.

Figure 4.2. O-type entities for describing lines.

value types	explanation
B-DVA	a base data-valued attribute
BS-DVA	a subcategory-defined base data-valued attribute
DI-DVA	an internally derived data-valued attribute
DE-DVA	an externally derived data-valued attribute
DS-DVA	a subcategory-defined derived data-valued attribute
B-OEVA	a base entity-valued attribute
BS-OEVA	a subcategory-defined base entity-valued attribute
DI-OEVA	an internally derived entity-valued attribute
DE-OEVA	an externally derived entity-valued attribute
DS-OEVA	a subcategory-defined derived entity-valued attribute

Figure 4.3. Different types of value types for O-type attributes.

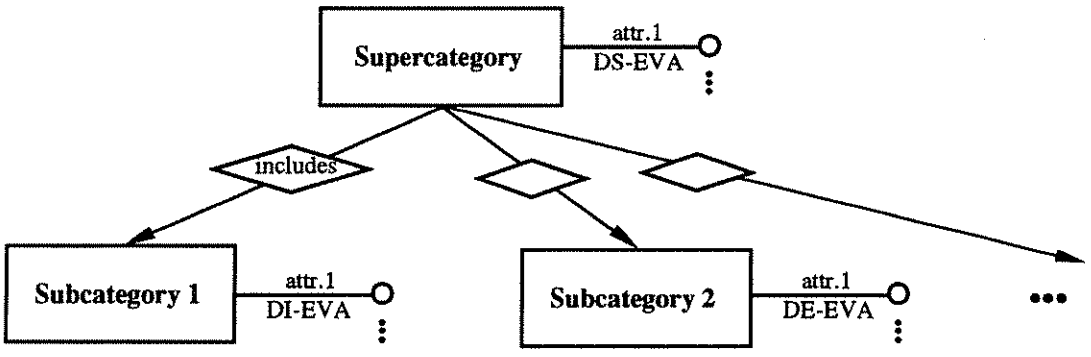
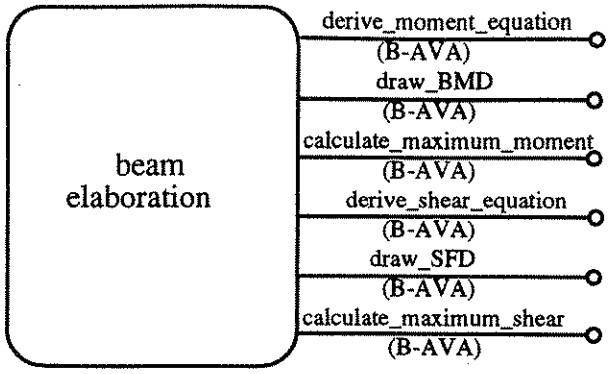
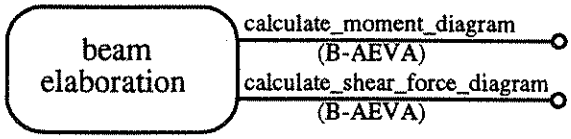


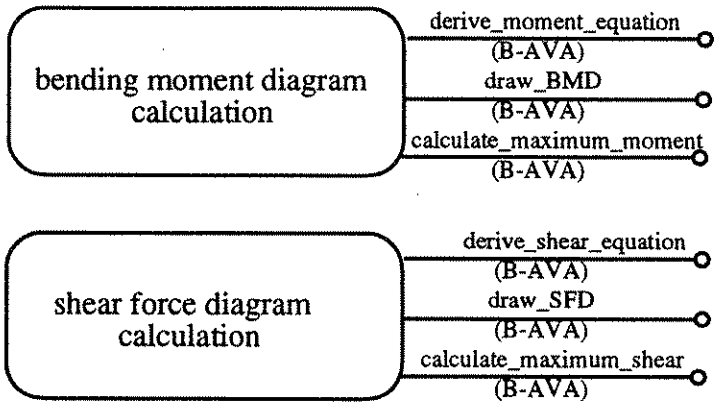
Figure 4.4. Value type specialization.



a. Beam elaboration entity with action-valued attributes.



b. Beam elaboration entity with entity-valued attributes.



c. Bending moment diagram calculation and shear force diagram calculation entities with action-valued attributes.

Figure 4.5. A-type entities for describing beam design elaboration activities.

value types	explanation
B-AVA	a base action-valued attribute
BS-AVA	a subcategory-defined base action-valued attribute
DI-AVA	an internally derived action-valued attribute
DE-AVA	an externally derived action-valued attribute
DS-AVA	a subcategory-defined derived action-valued attribute
B-AEVA	a base entity-valued attribute
BS-AEVA	a subcategory-defined base entity-valued attribute
DI-AEVA	an internally derived entity-valued attribute
DE-AEVA	an externally derived entity-valued attribute
DS-AEVA	a subcategory-defined derived entity-valued attribute

Figure 4.6. Different types of value types for A-type attributes.

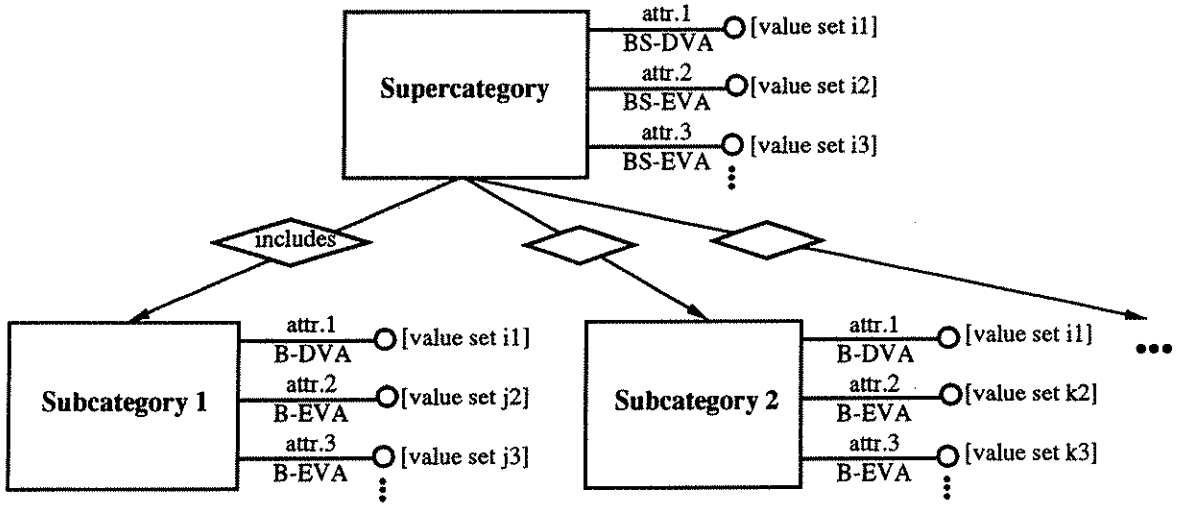


Figure 4.7. Value set specialization.

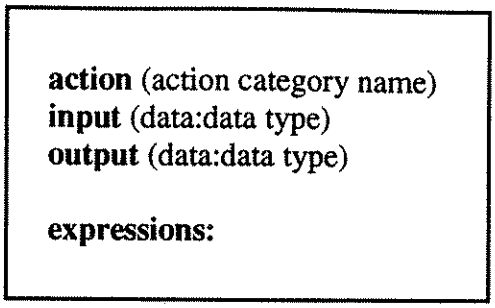
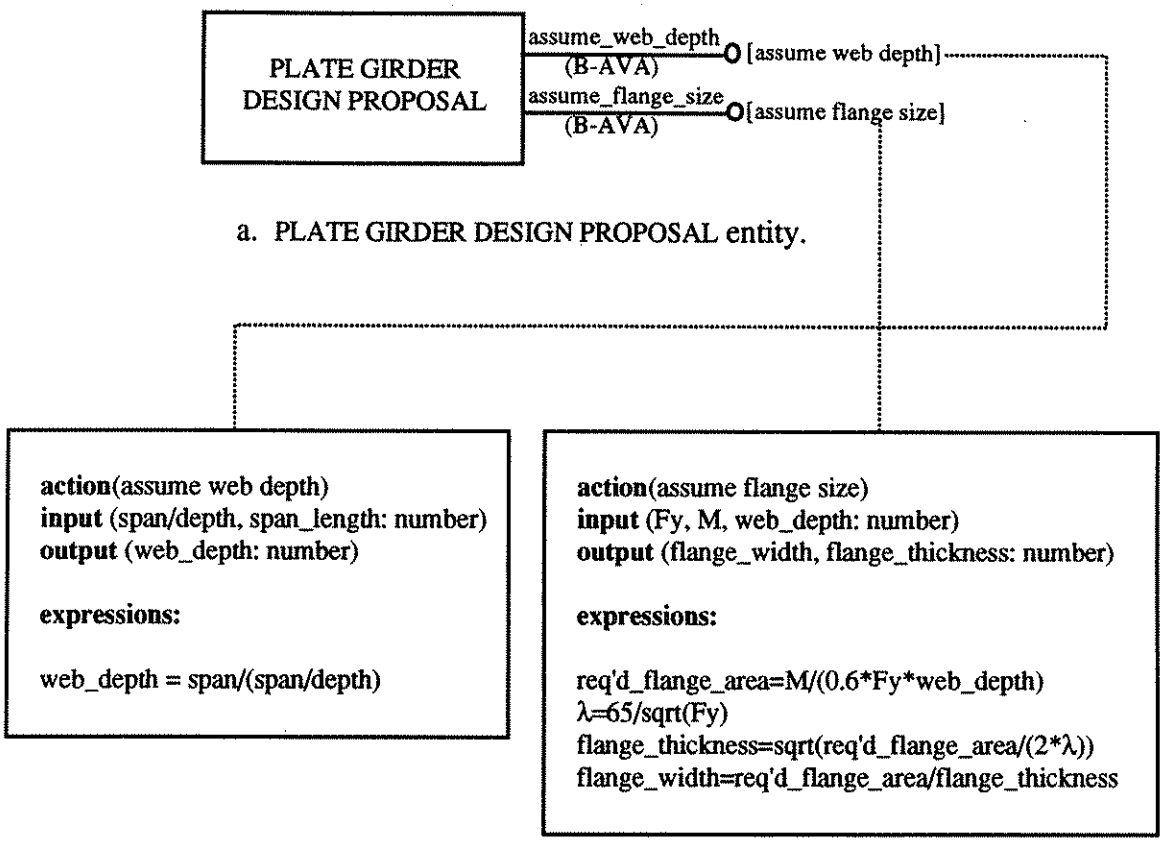


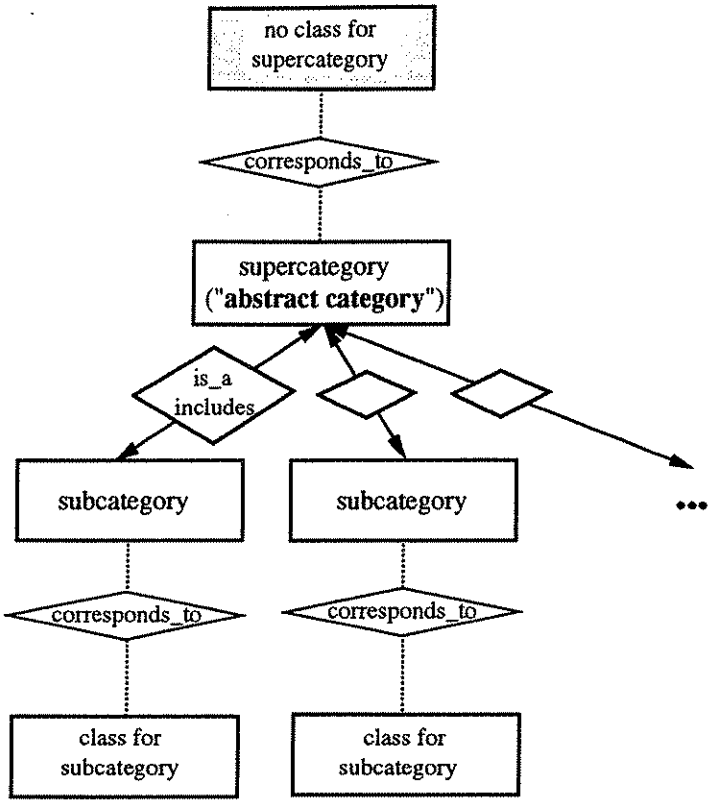
Figure 4.8. Description of an action category.



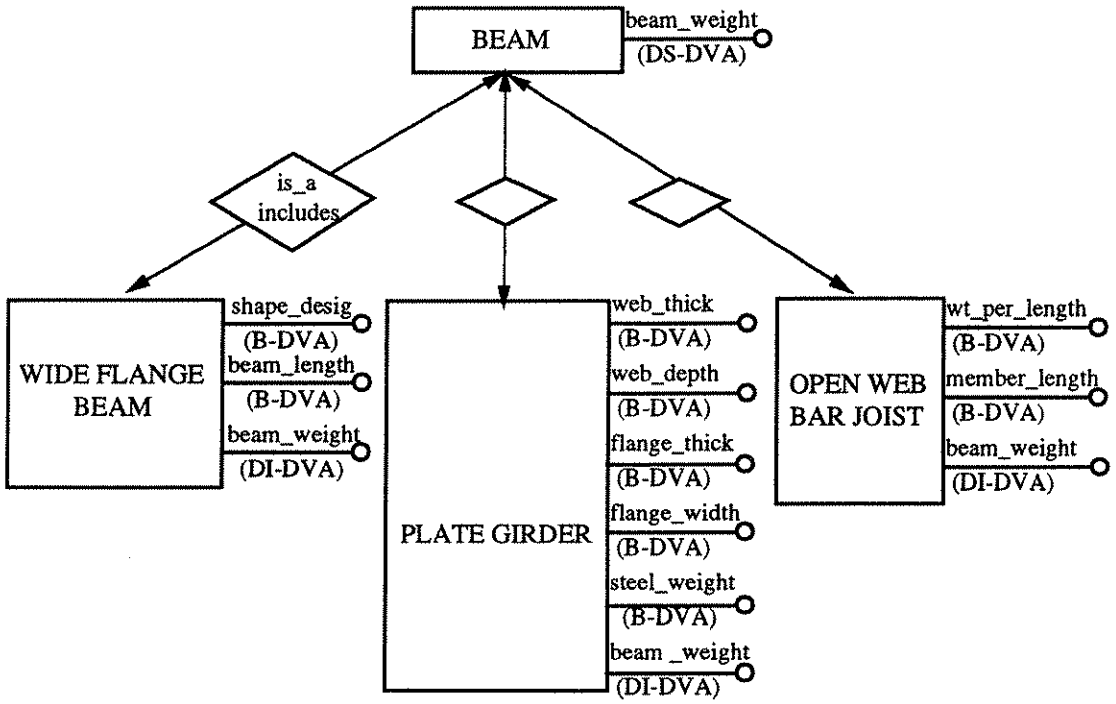
a. PLATE GIRDER DESIGN PROPOSAL entity.

b. Actions defined for the value sets of a PLATE GIRDER DESIGN PROPOSAL category.

Figure 4.9. Examples of action categories.

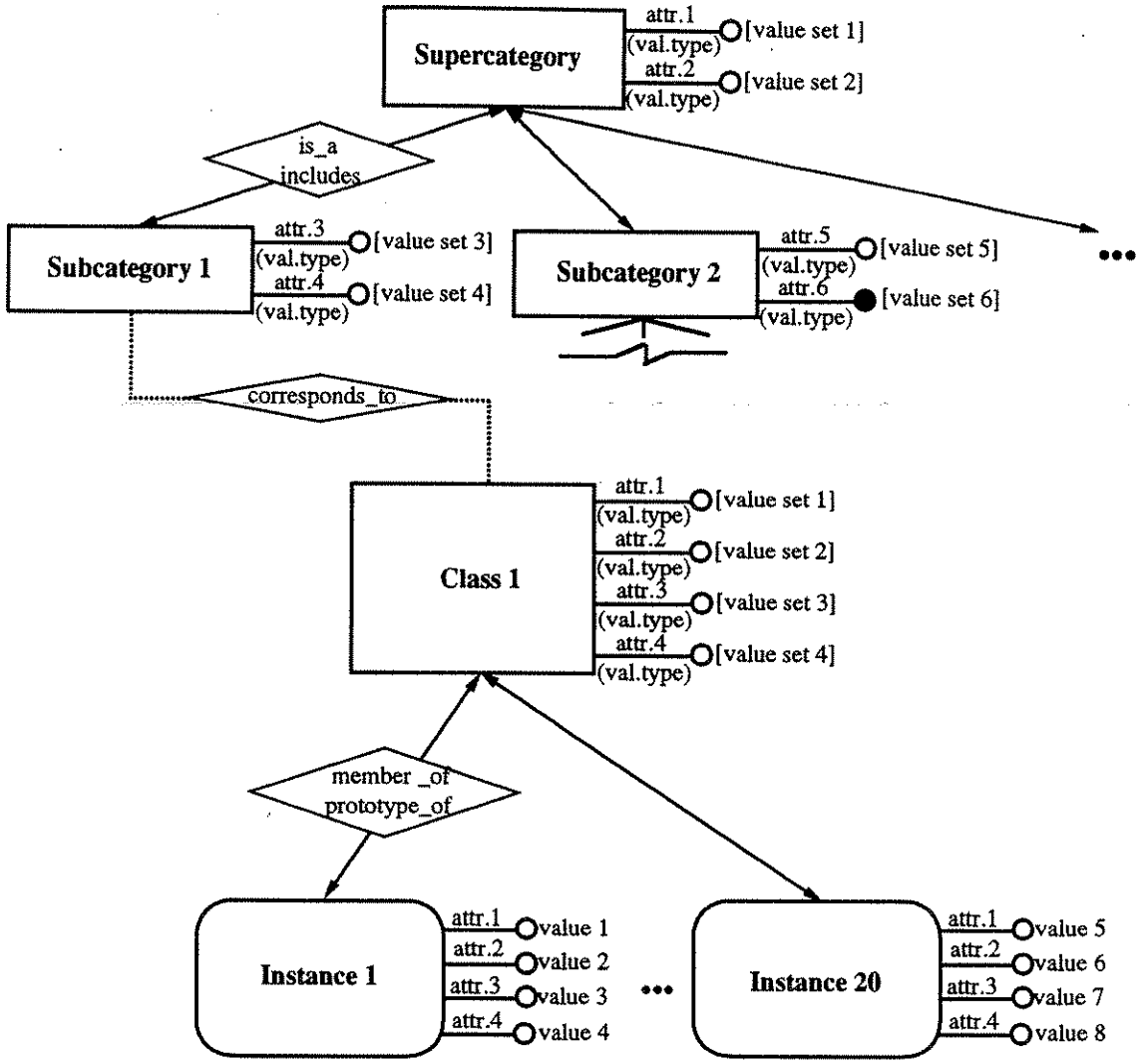


a. Generalization tree with an abstract category.

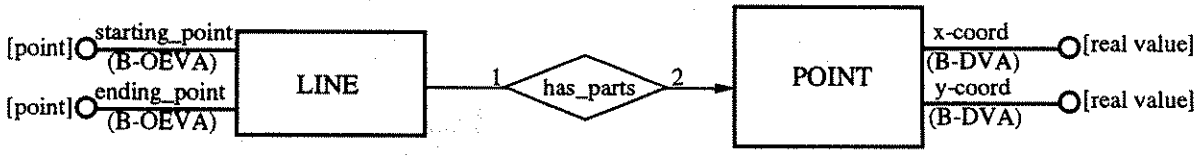


b. Example of generalization tree with an abstract category.

Figure 4.10. Abstract categories.



a. General modified E-R diagram.



b. Examples of the modified E-R diagram.

Figure 4.11. Modified E-R diagram.

entity category name:			
supercategory name:			
attribute name	value set	value type	cardinality

a. Table representation for an entity category.

instance name:			
class name:			
attribute name	value	value type	cardinality

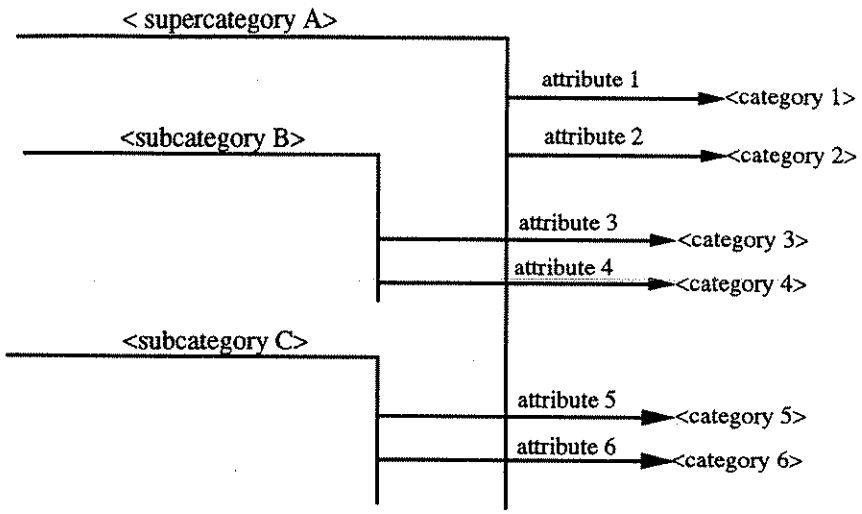
b. Table representation for an instance.

entity category name: line			
supercategory name: none			
attribute name	value set	value type	cardinality
starting_point ending_point	point point	B-OEVA B-OEVA	SV SV

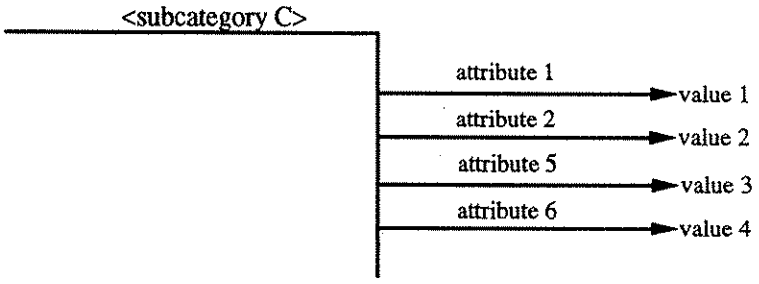
entity category name: point			
supercategory name: none			
attribute name	value set	value type	cardinality
x-coord y-coord	real number real number	B-DVA B-DVA	SV SV

c. Example of table representation.

Figure 4.12. Table representation.

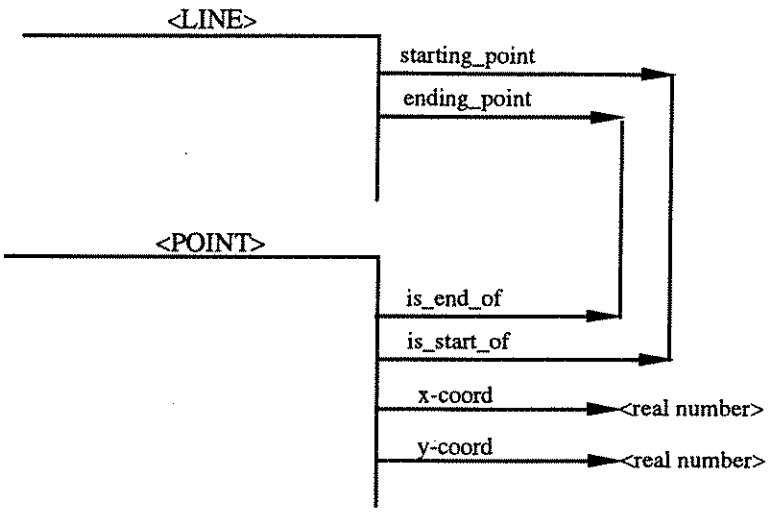


a. General SDM diagram.

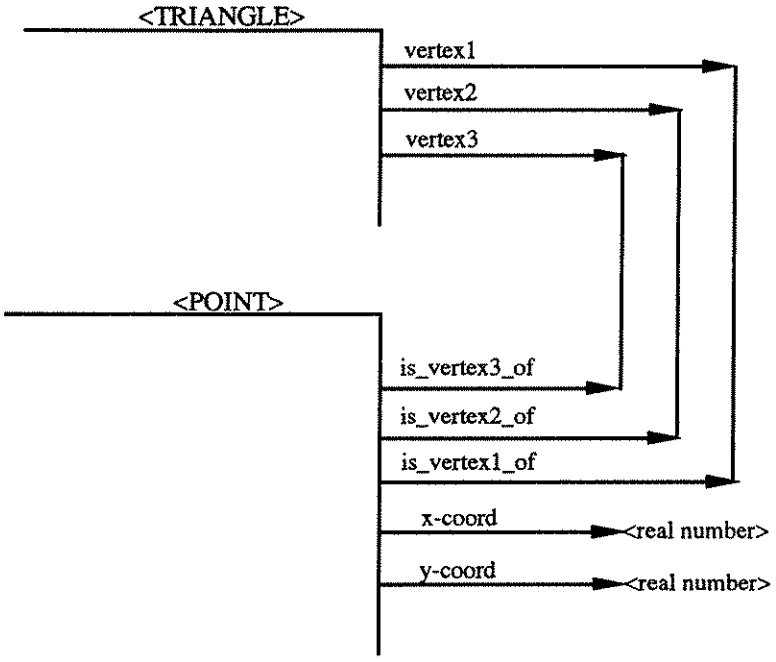


b. SDM diagram for an instance.

Figure 4.13. SDM diagrams.

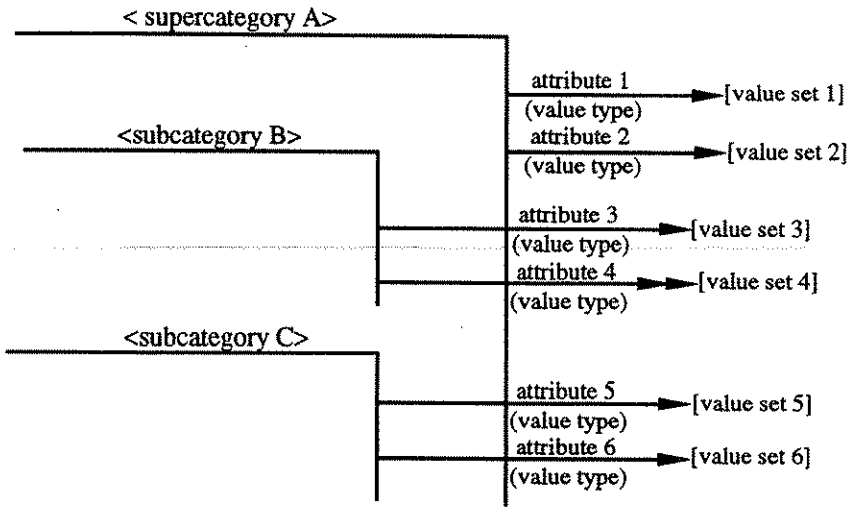


c. SDM diagram for LINE category.

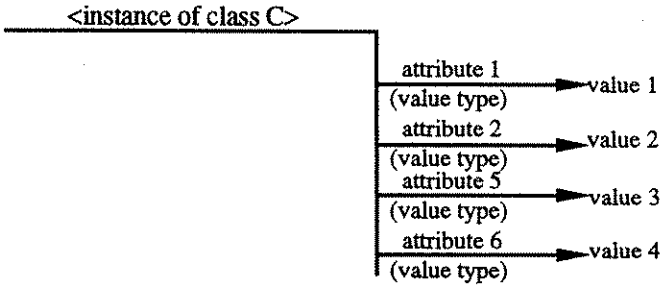


d. SDM diagram for TRIANGLE category.

Figure 4.13. SDM diagrams (continued).

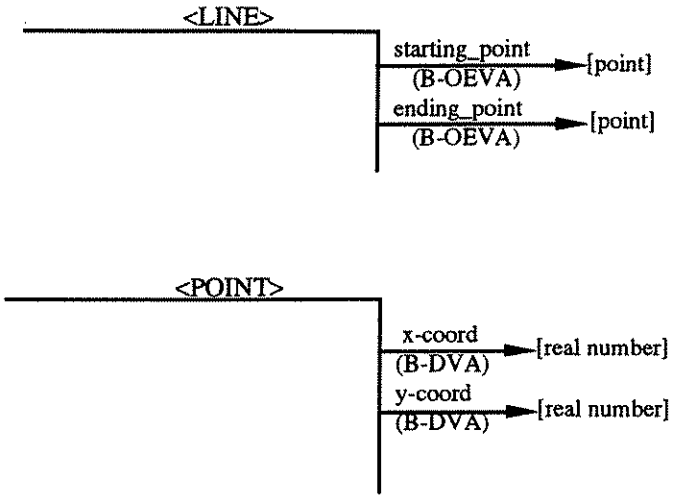


a. General modified SDM diagram.

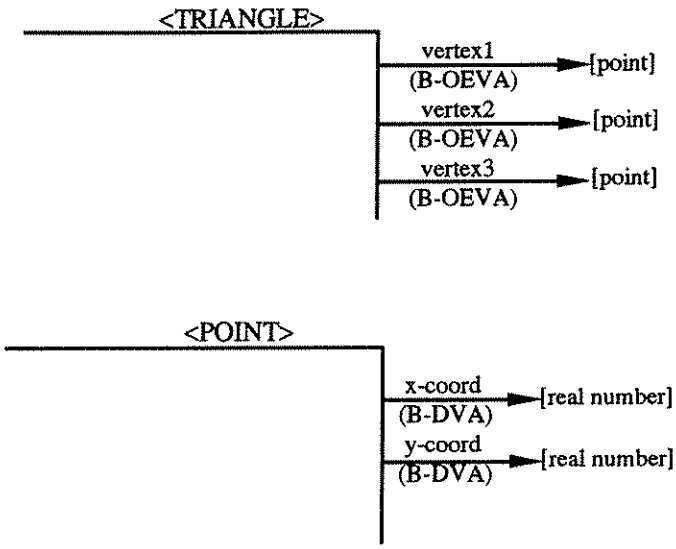


b. Modified SDM diagram for an instance.

Figure 4.14. Modified SDM diagrams.



c. Modified SDM diagram for LINE category.



d. Modified SDM diagram for TRIANGLE category.

Figure 4.14. Modified SDM diagrams (continued).

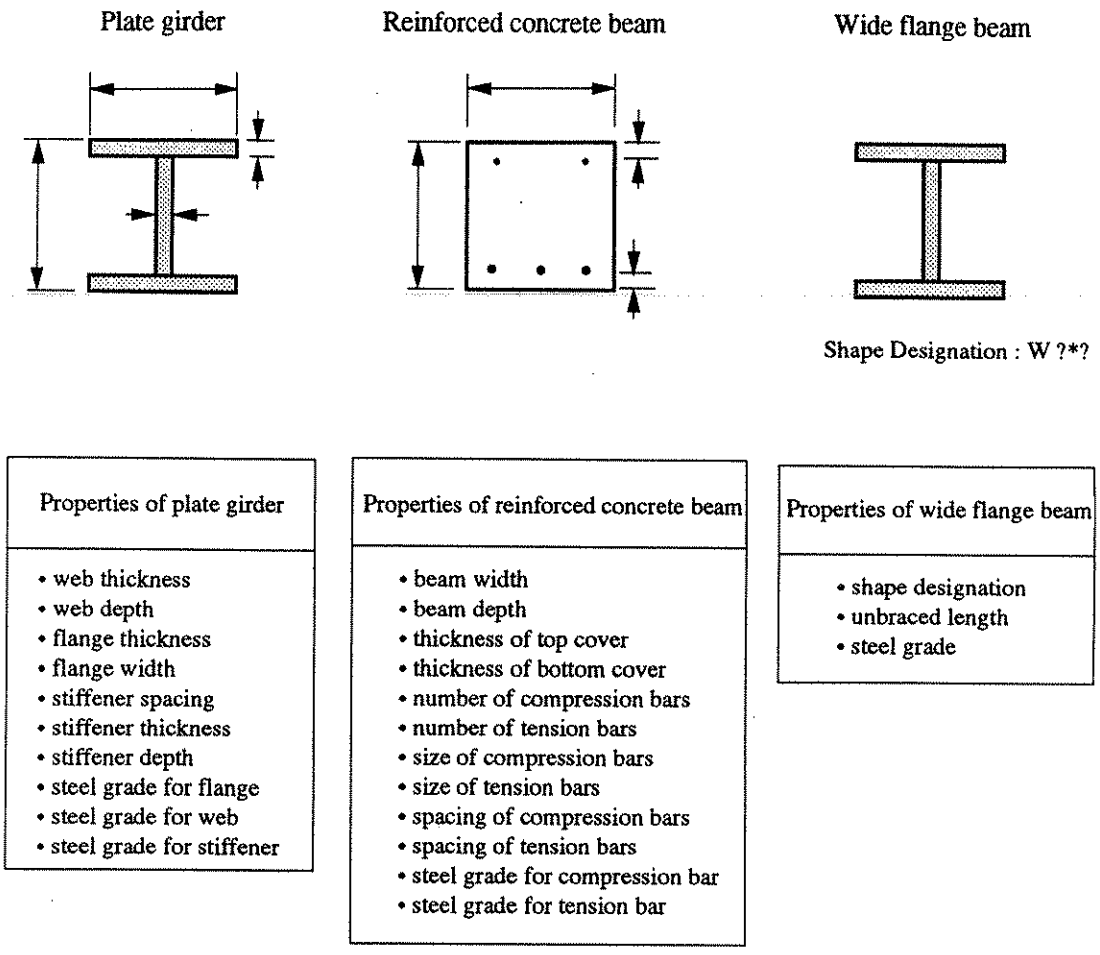


Figure 5.1. Properties of different beam design alternatives.

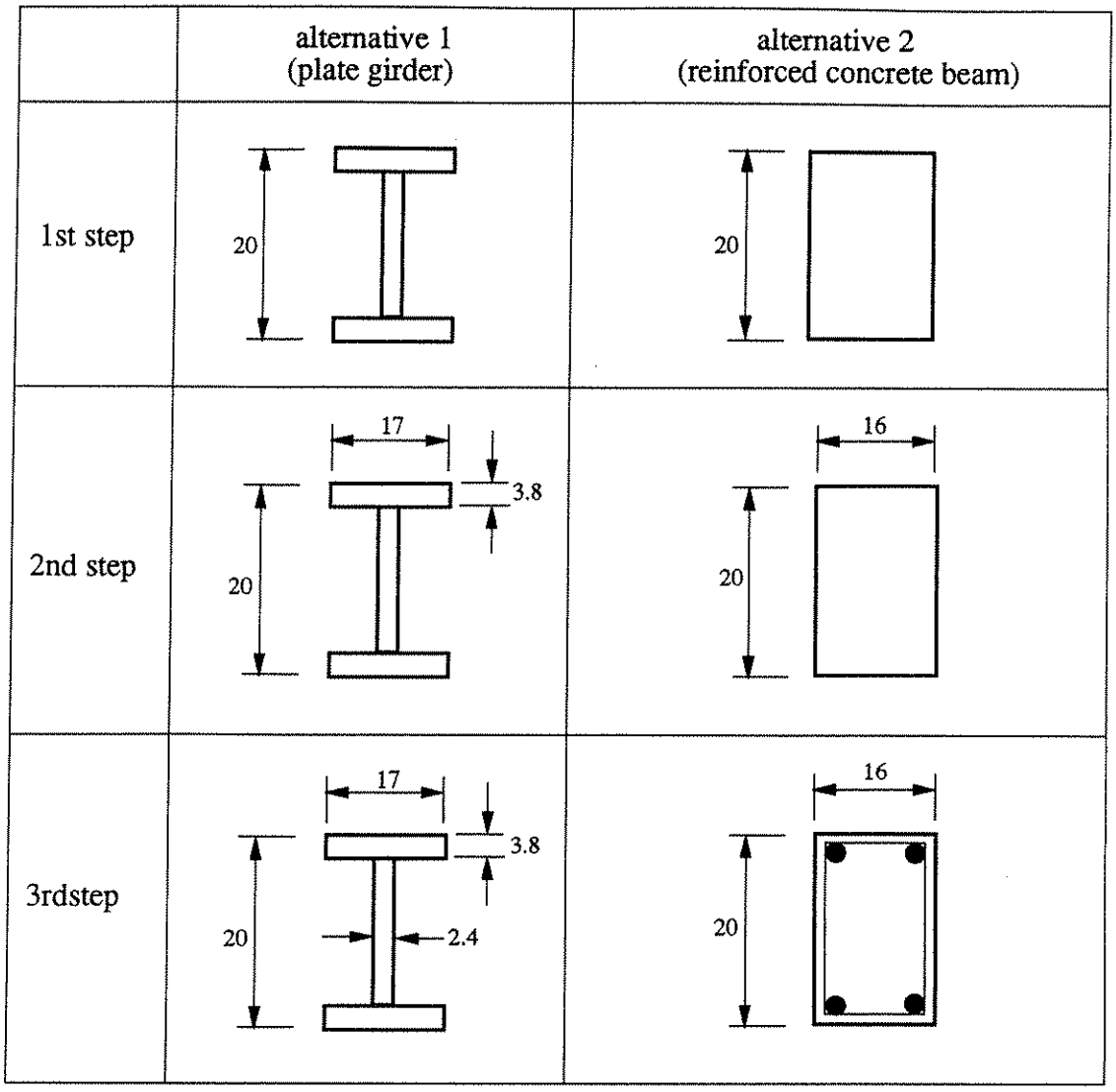
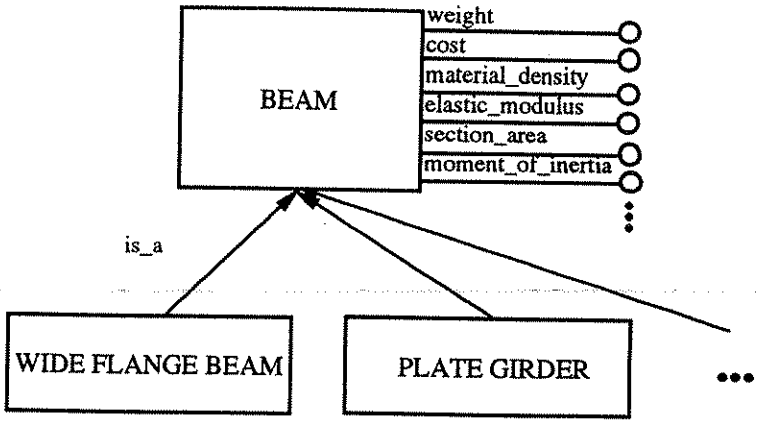
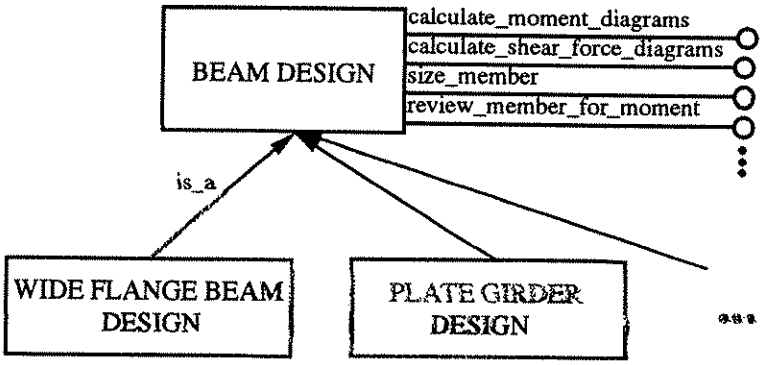


Figure 5.2. Data growth for different design alternatives.

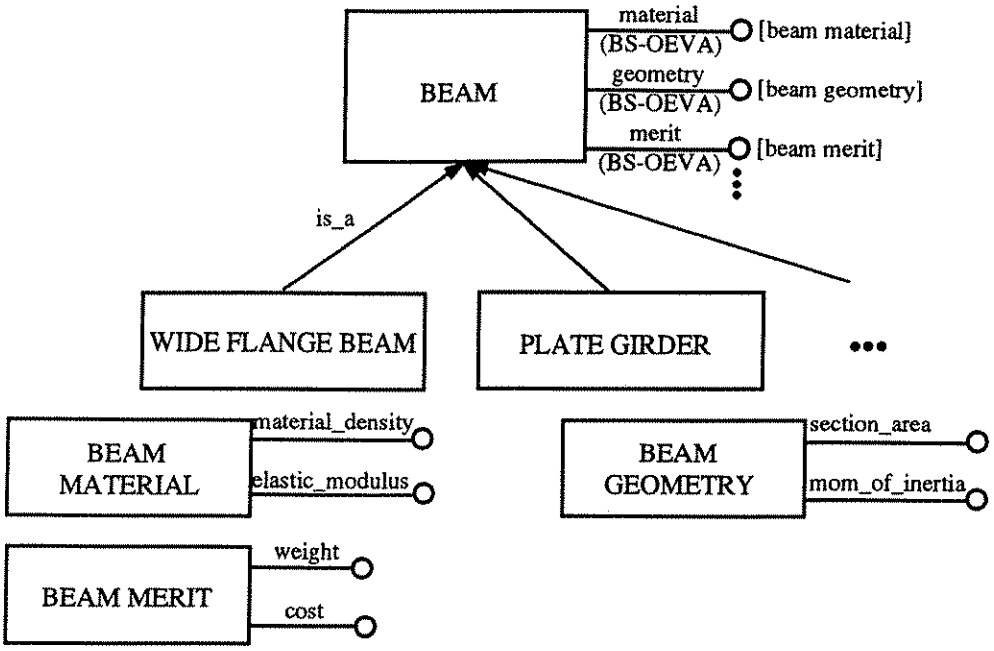


a. Generalization of design alternatives for product model.

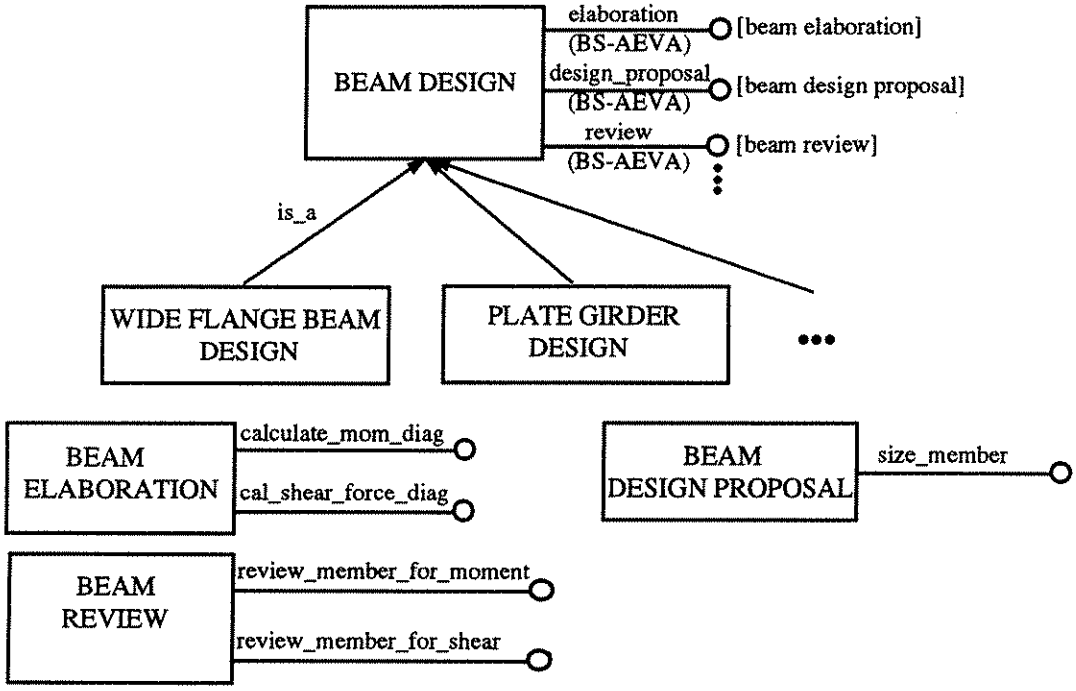


b. Generalization of design alternatives for process model.

Figure 5.3. Examples of generalization abstraction.

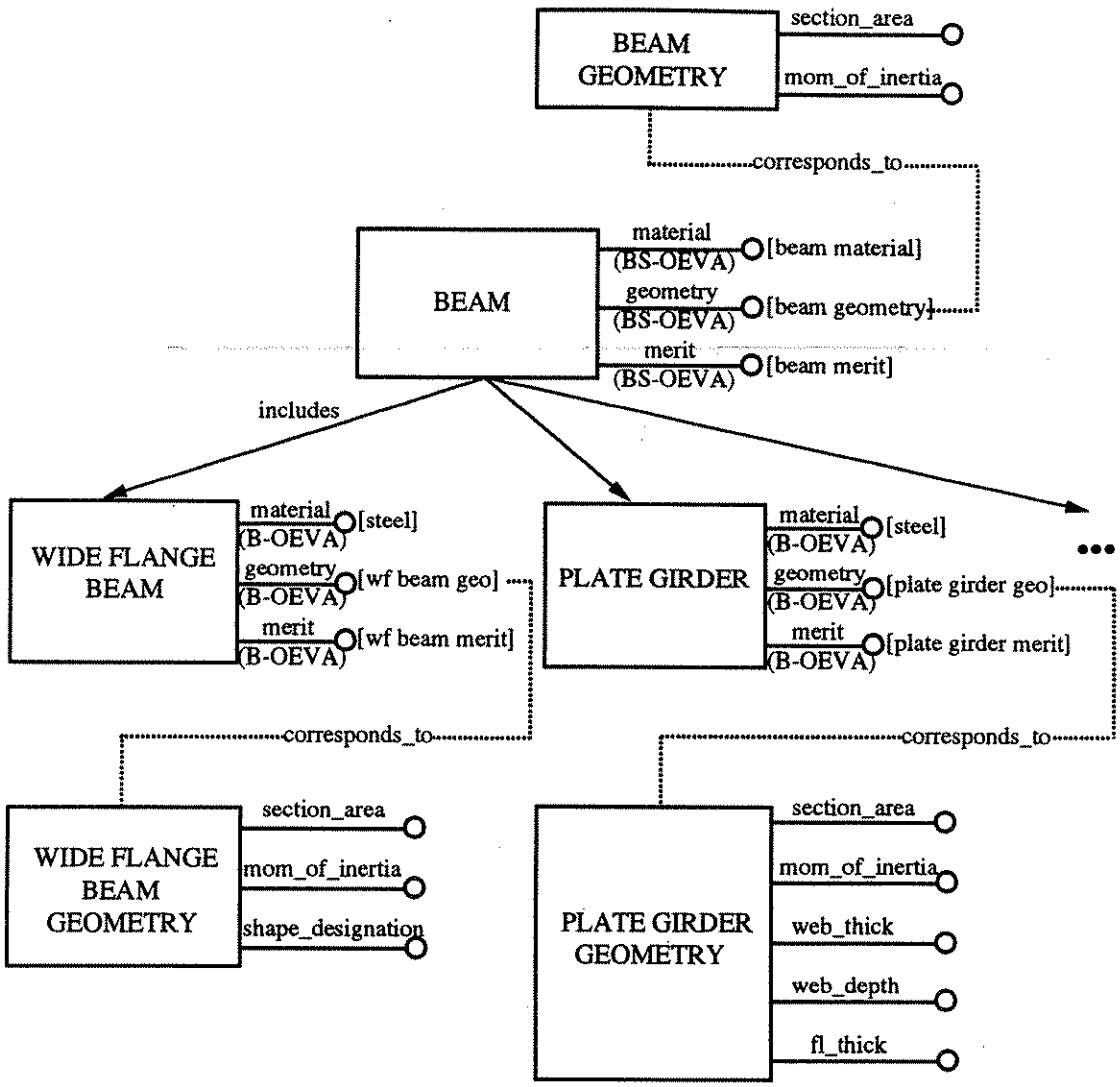


a. Generalization of design alternatives for product model.



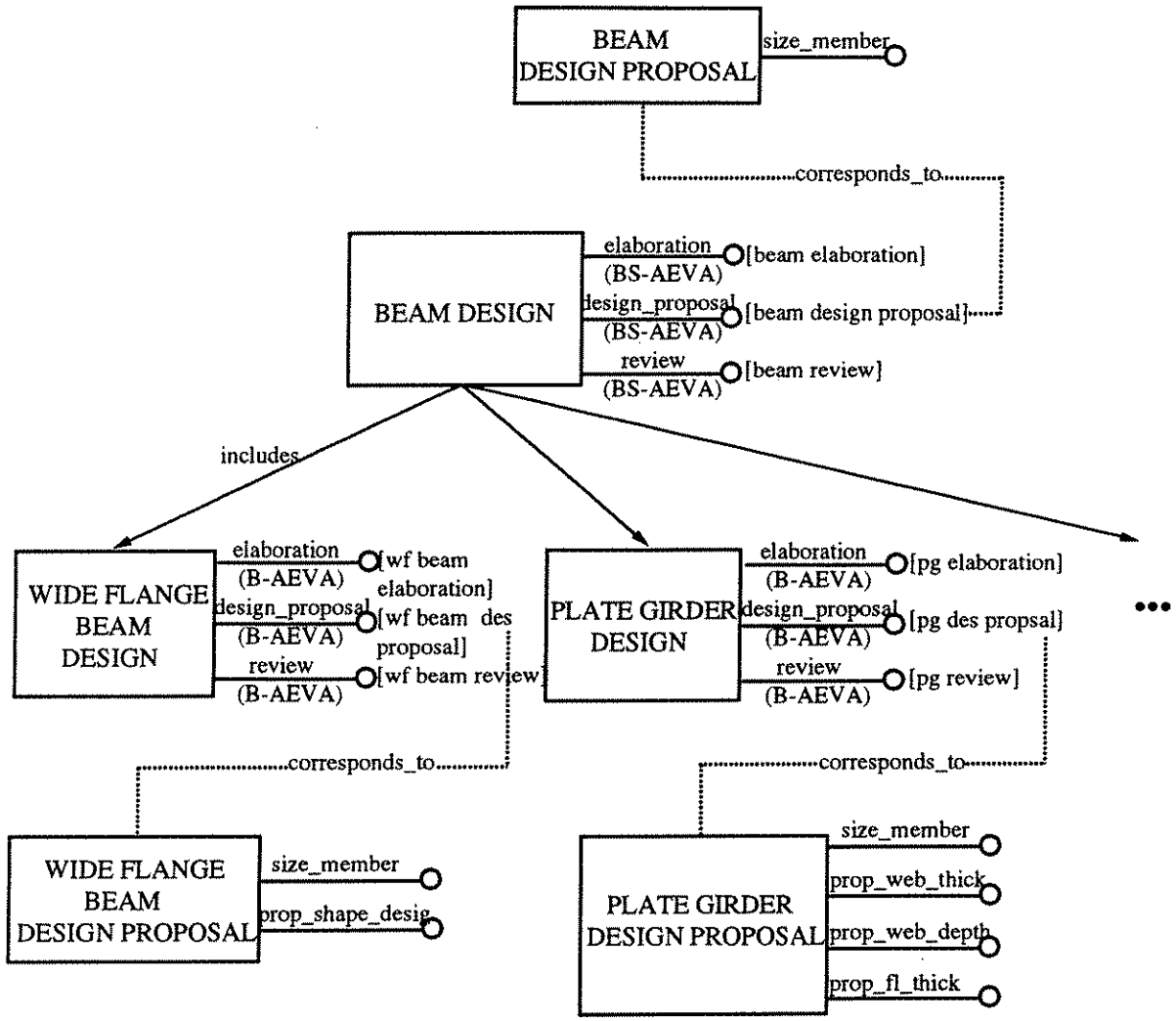
b. Generalization of design activities for process model.

Figure 5.4. Examples of generalization abstraction of design alternatives using entity-valued attributes.



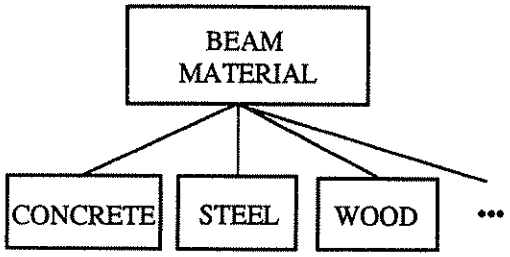
a. Specialization of value sets for product model.

Figure 5.5. Specialization of value sets.

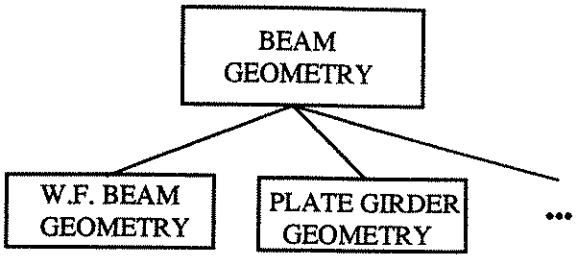


b. Specialization of value sets for process model.

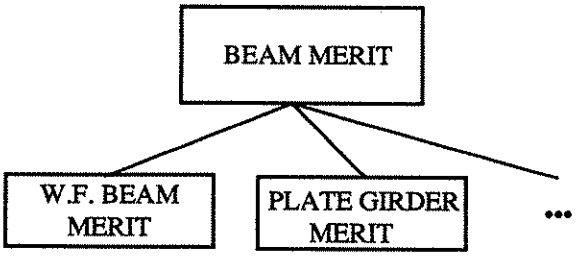
Figure 5.5. Specialization of value sets (continued).



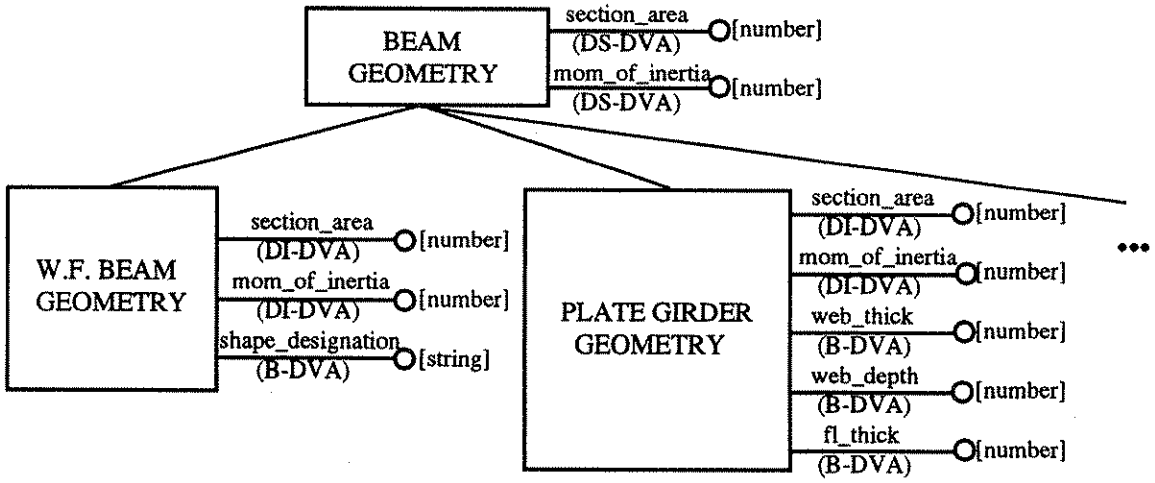
a. Beam material hierarchy.



b. Beam geometry hierarchy.

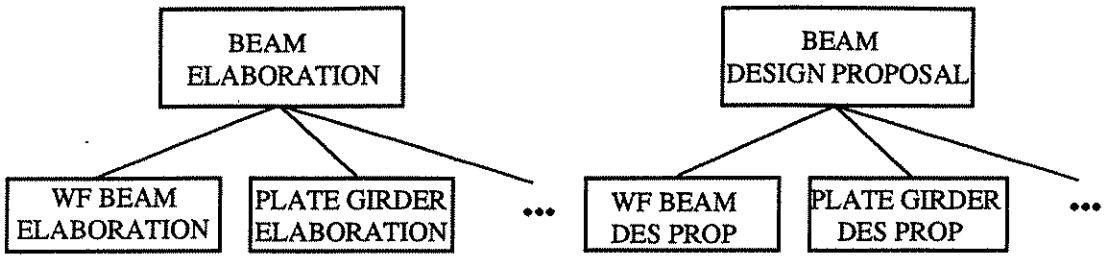


c. Beam merit hierarchy.



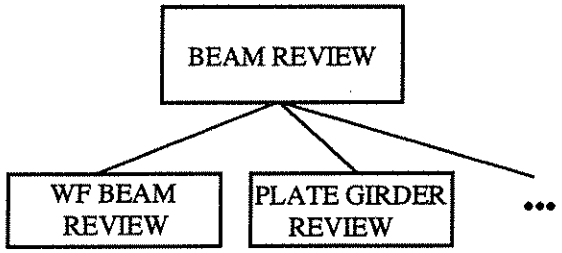
d. Beam geometry hierarchy with attributes.

Figure 5.6. Examples of O-type primitive hierarchies for product model.

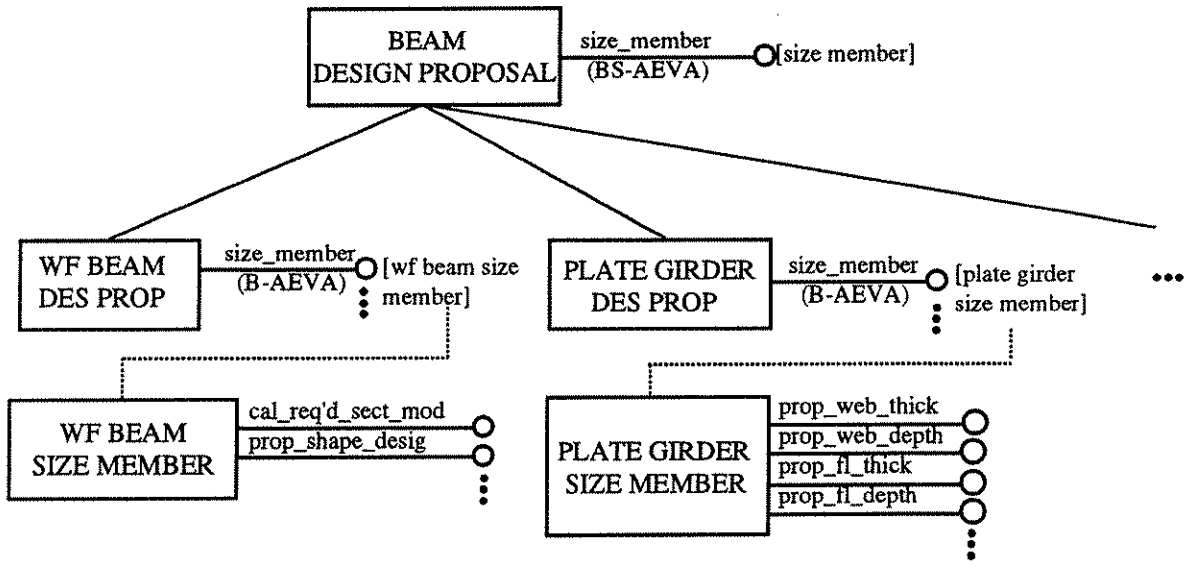


a. Beam elaboration hierarchy.

b. Beam design proposal hierarchy.



c. Beam review hierarchy.



d. Beam design proposal hierarchy with attributes.

Figure 5.7. Examples of A-type primitive hierarchies for process model.

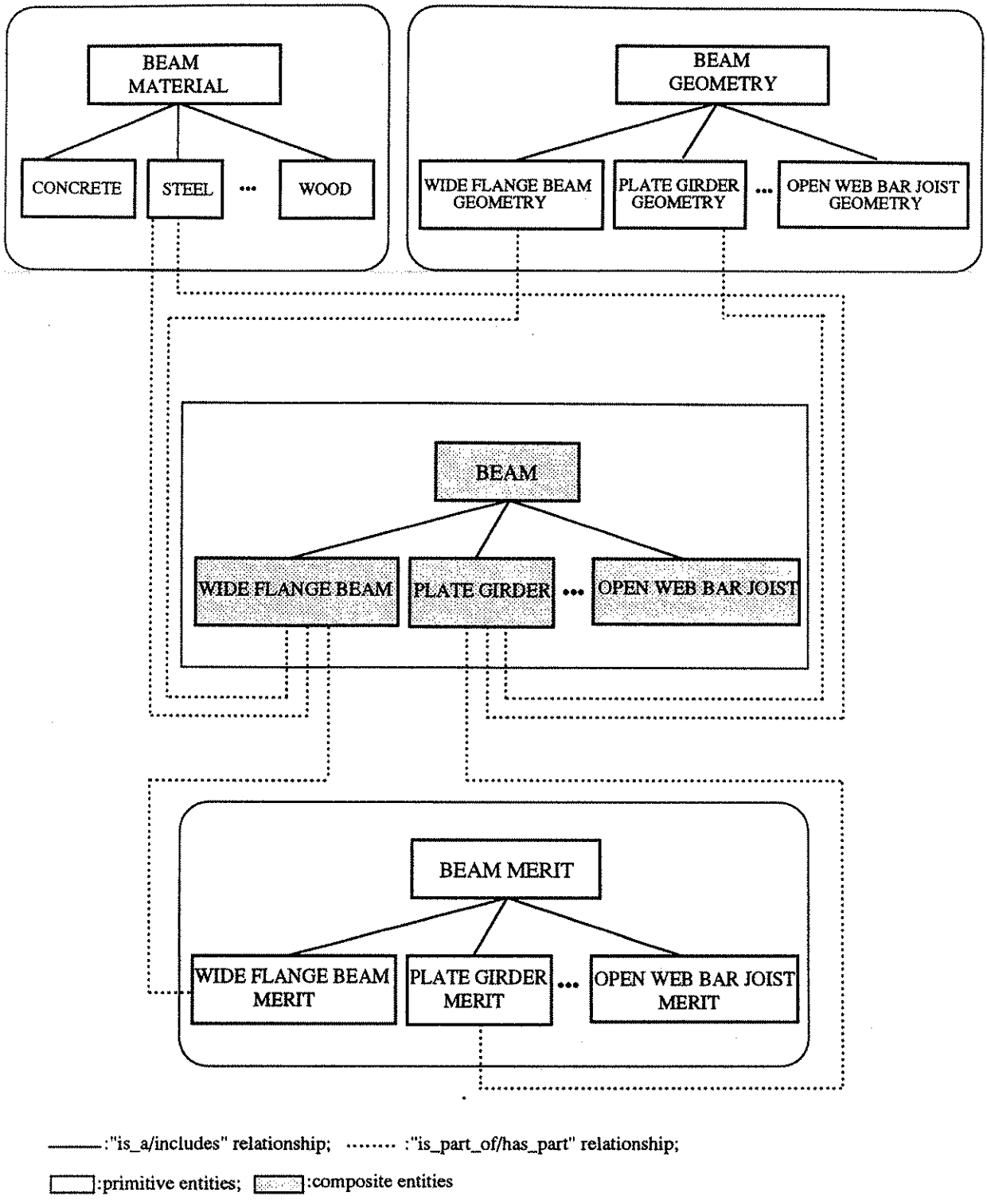


Figure 5.8. O-type composite entities defined by aggregating O-type primitive entities.

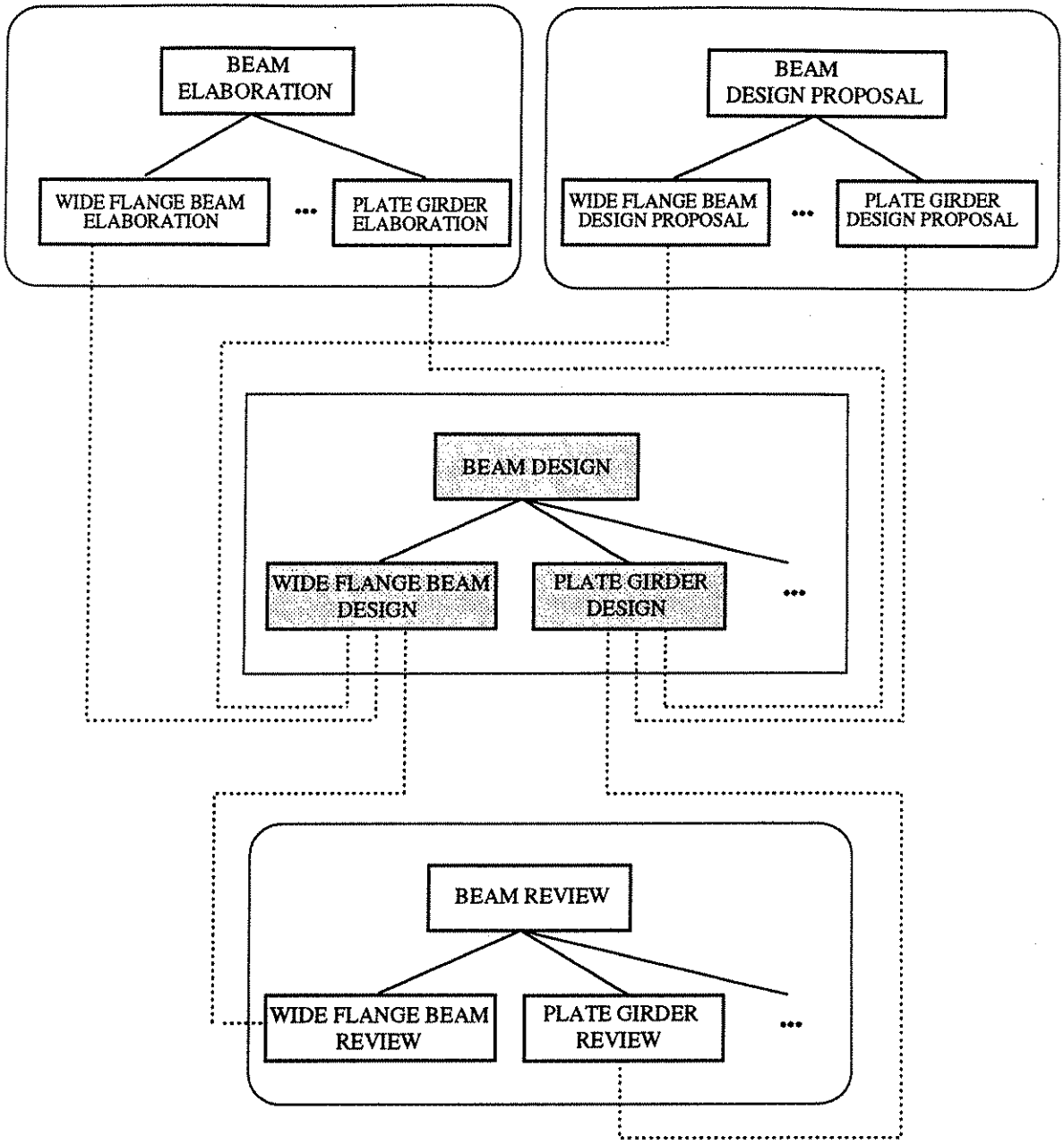


Figure 5.9. A-type composite entities defined by aggregating A-type primitive entities.

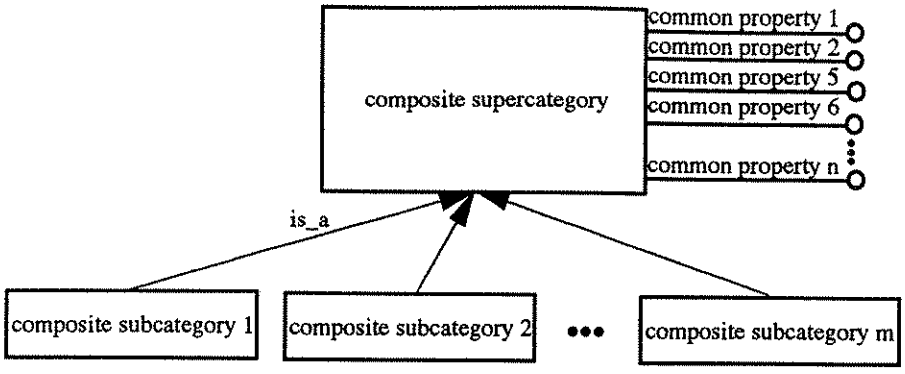


Figure 5.10. Composite generalization hierarchy for design alternatives.

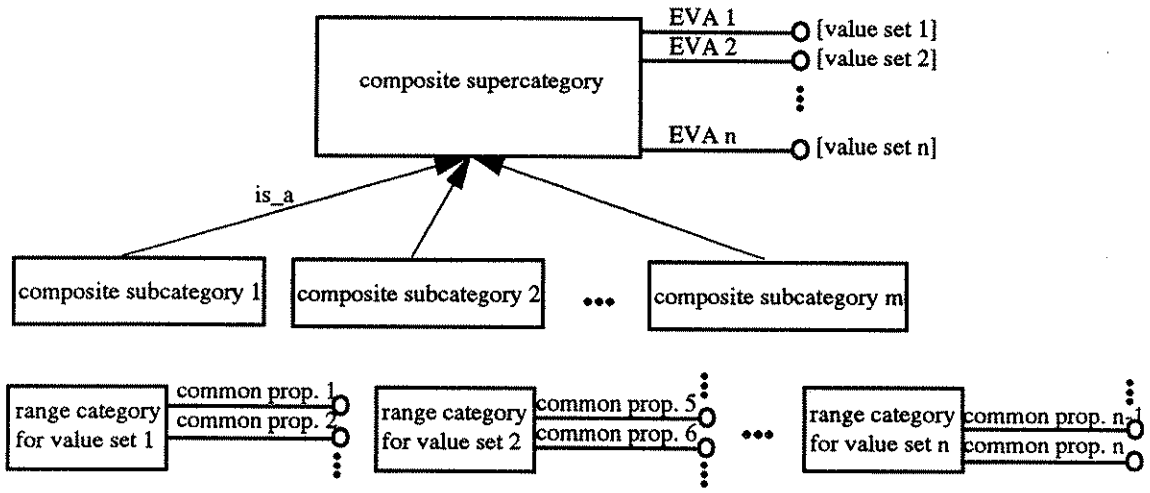


Figure 5.11. Modified composite generalization hierarchy for design alternatives.

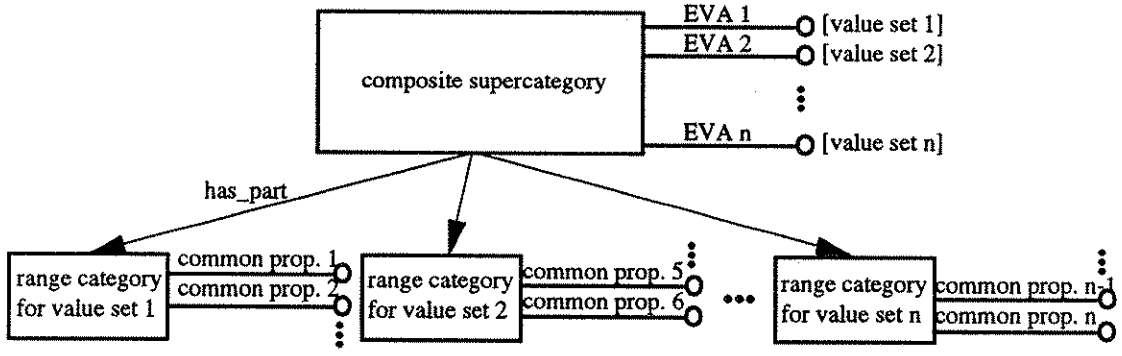
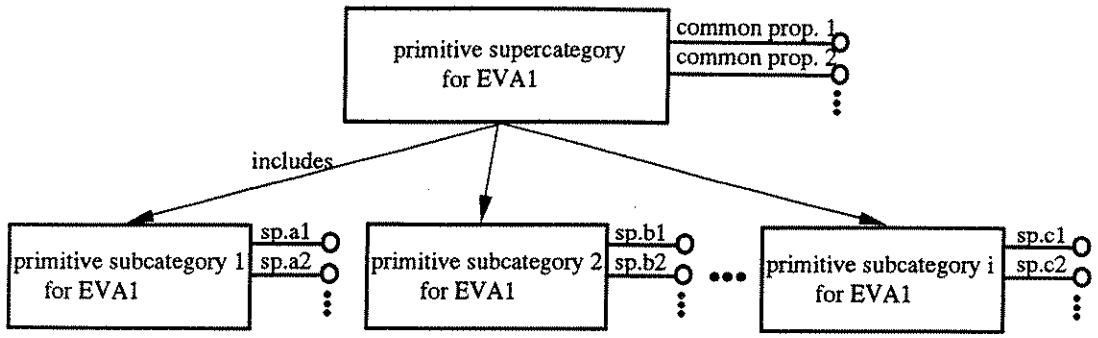
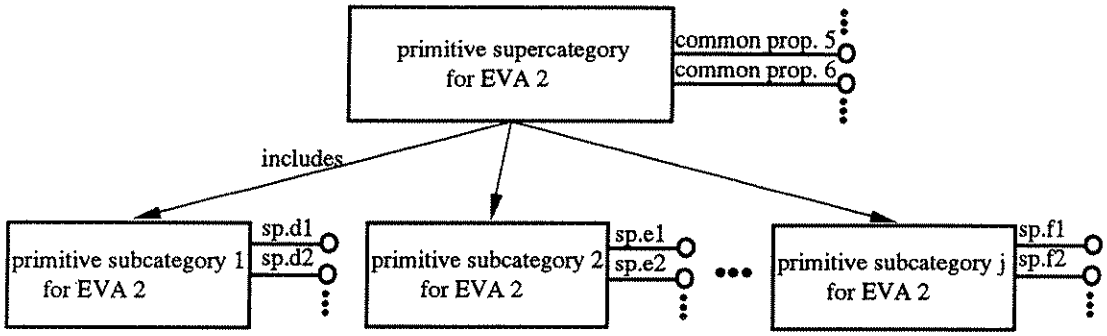


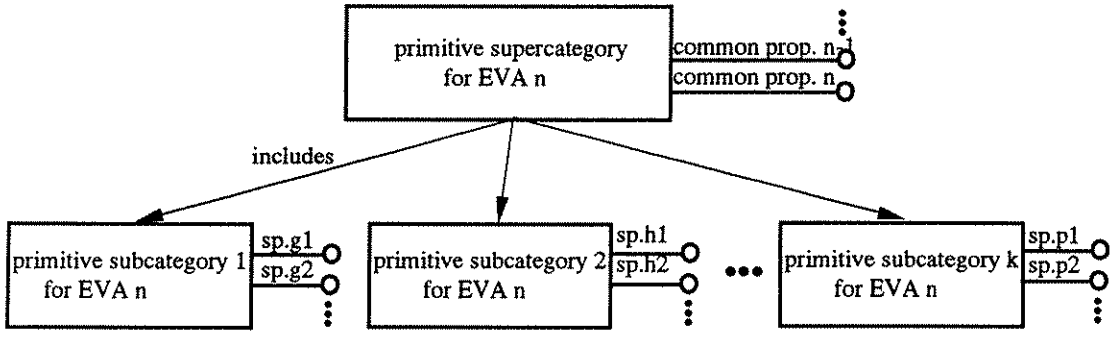
Figure 5.12. Decomposition of composite supercategory.



a. Primitive hierarchy for EVA 1.



b. Primitive hierarchy for EVA 2.



c. Primitive hierarchy for EVA n.

Figure 5.13. Primitive generalization hierarchies for design alternatives.

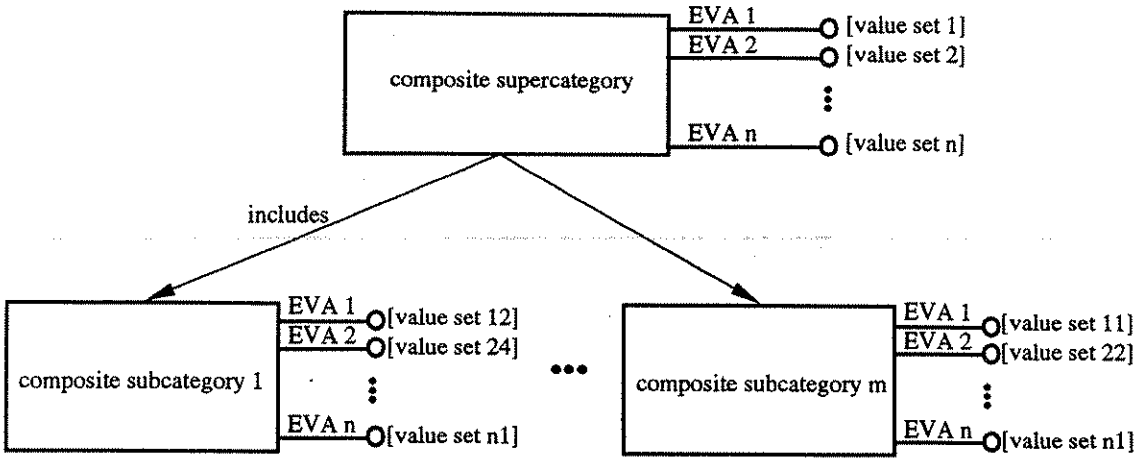


Figure 5. 14. Specialization of value sets.

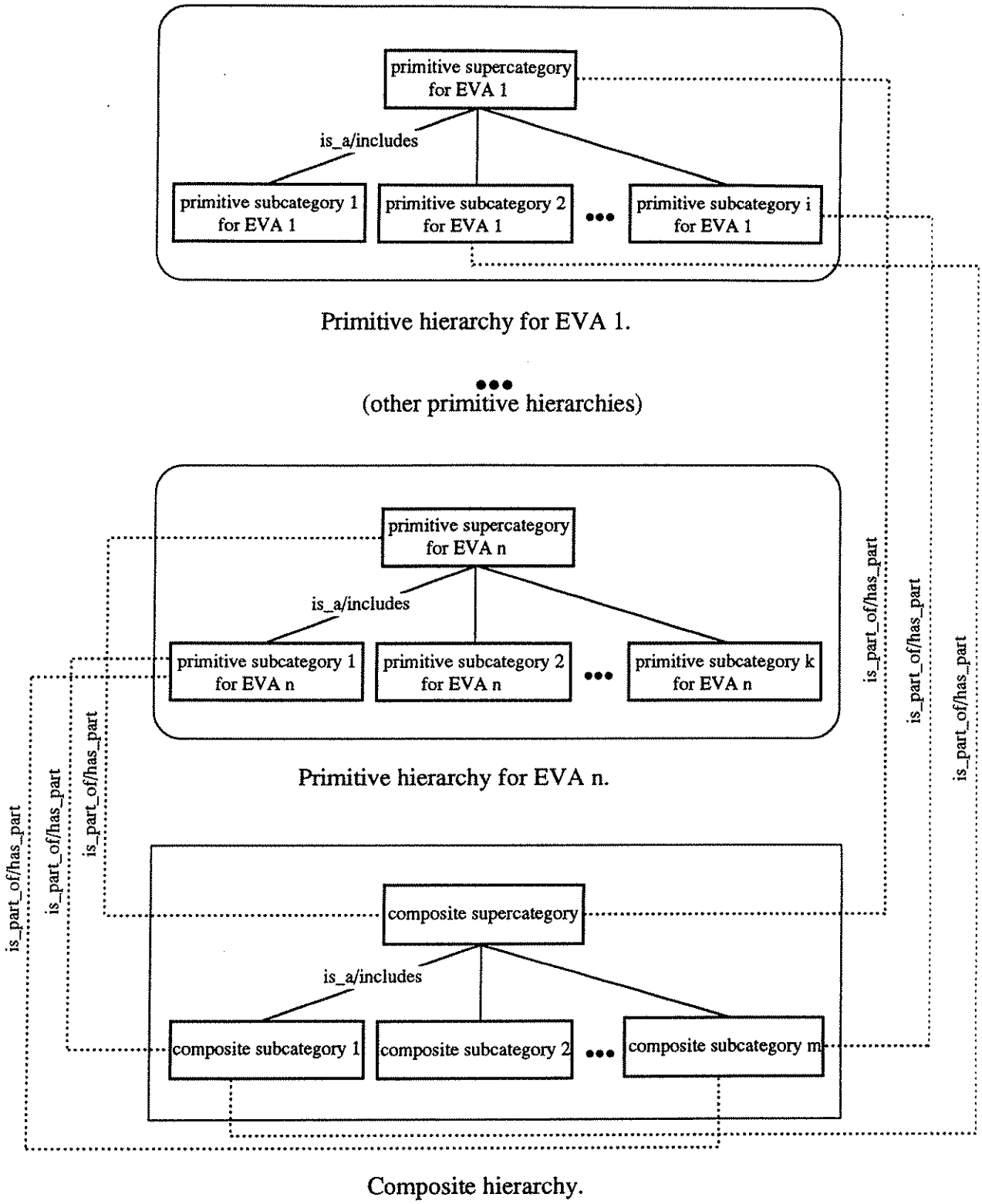


Figure 5.15. Relationships between primitive and composite hierarchies.

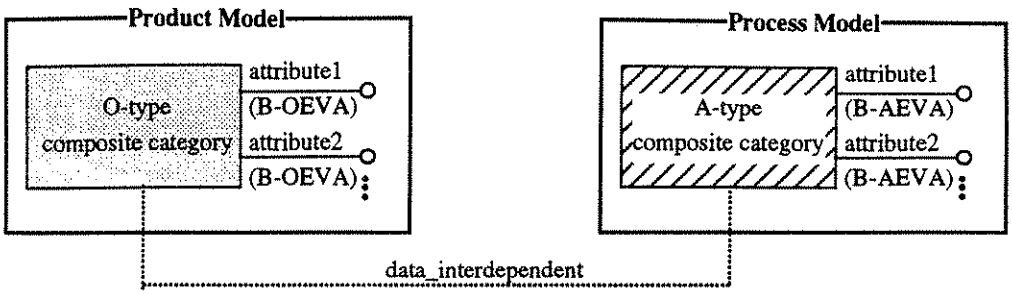


Figure 5.16. Data_interdependent relationship between O-type and A-type composite categories.

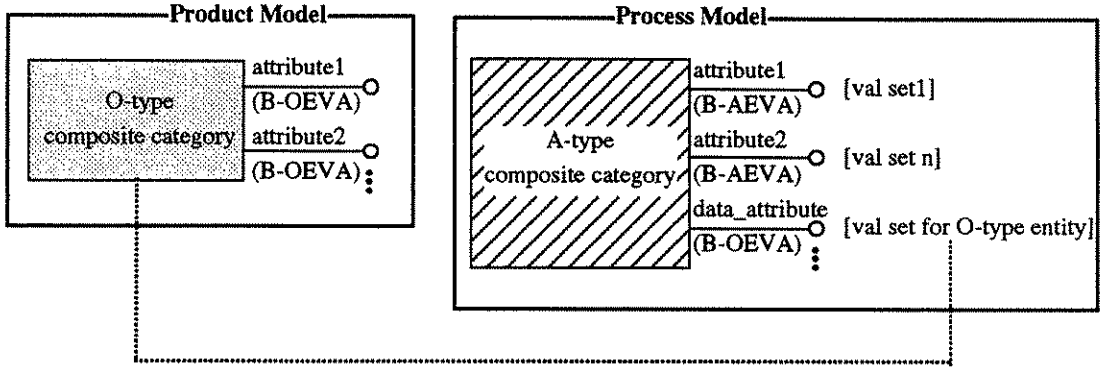


Figure 5.17. Formalization of the data_interdependent relationship as a data attribute.

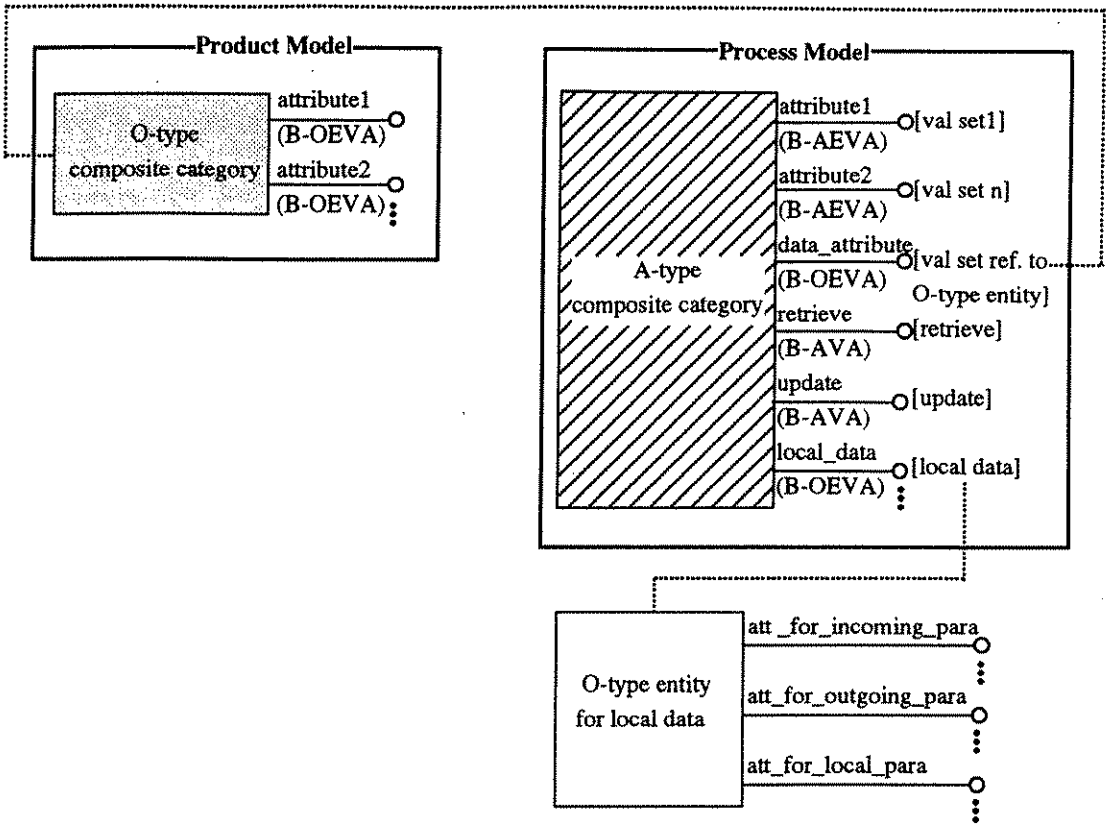


Figure 5.18. Creation of an O-type entity category for local data.

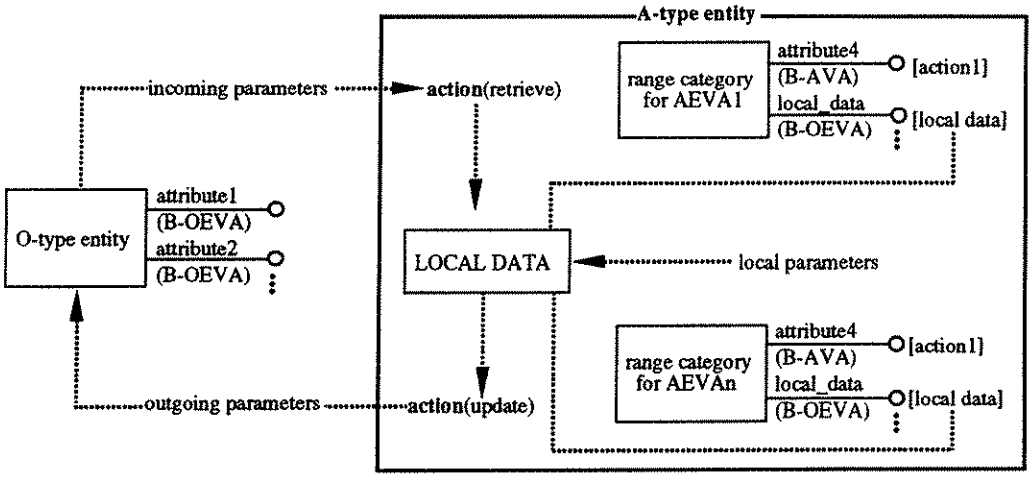
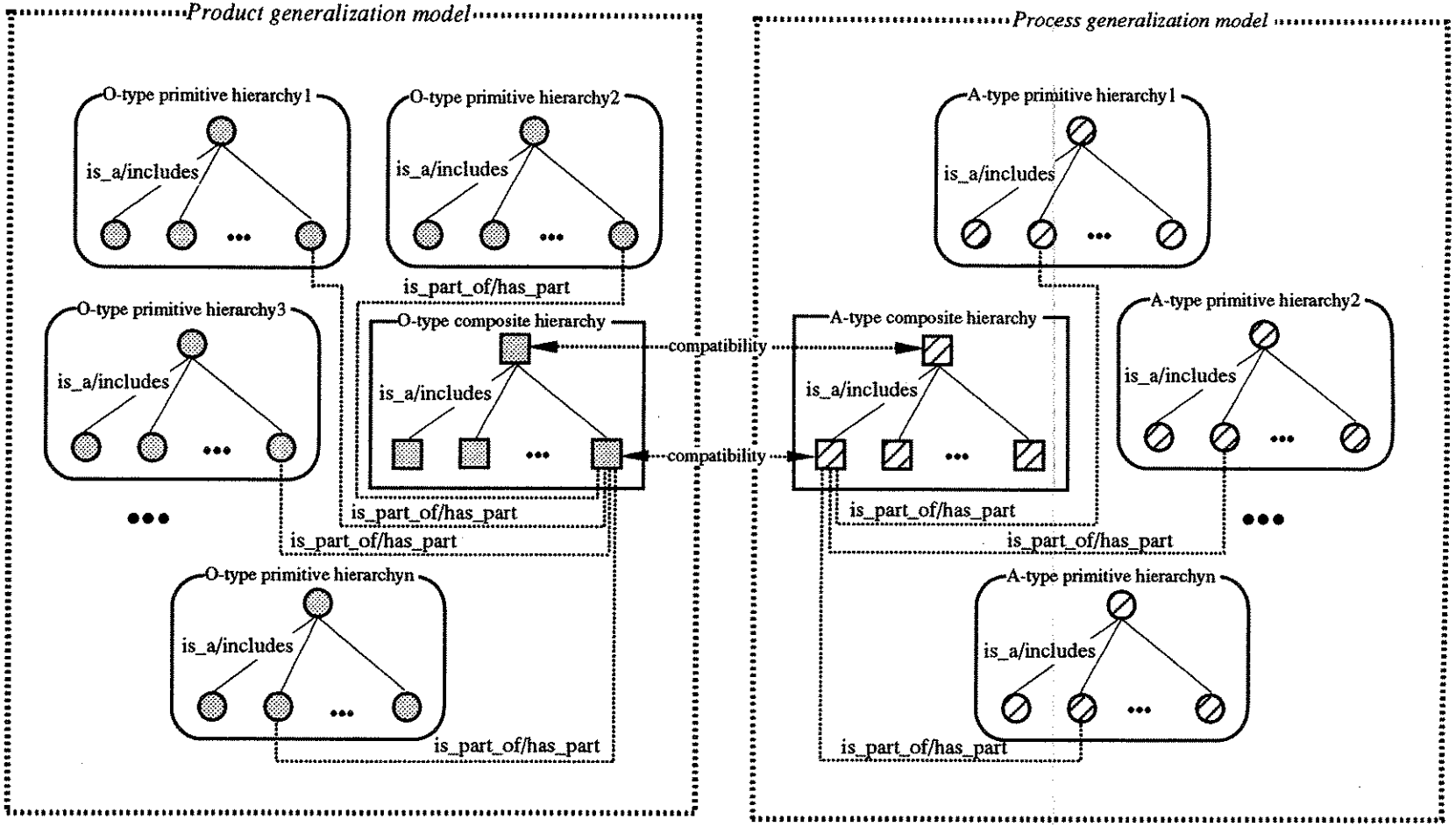


Figure 5.19. Retrieve, update, and local_data attributes of A-type entity.



a. Product generalization model.

b. Process generalization model.

Figure 5.20. Compatibility between O-type and A-type composite categories.

: O-type composite entity category : O-type primitive entity category : A-type composite entity category : A-type primitive entity category

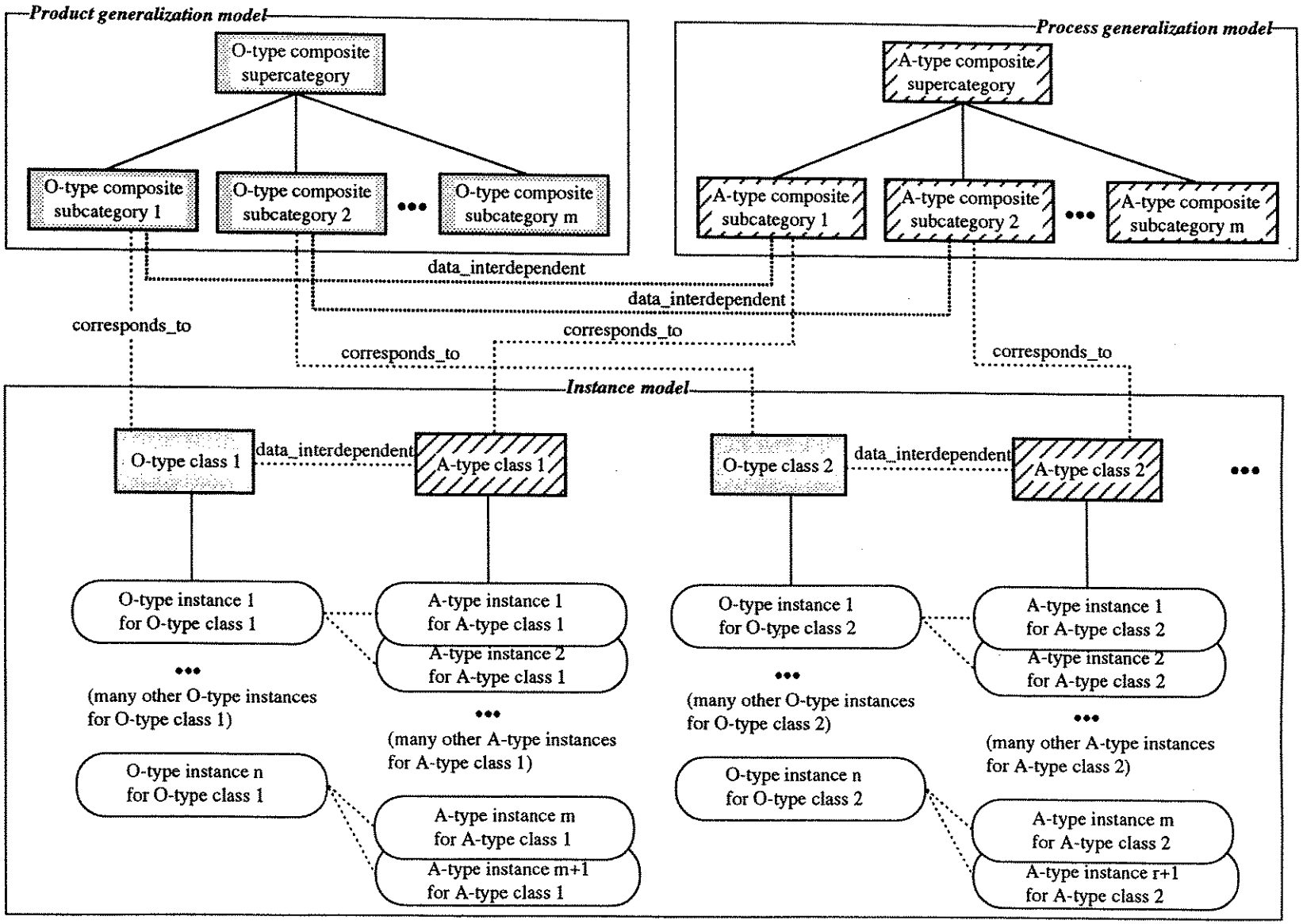


Figure 5.21. Creation of an instance model.

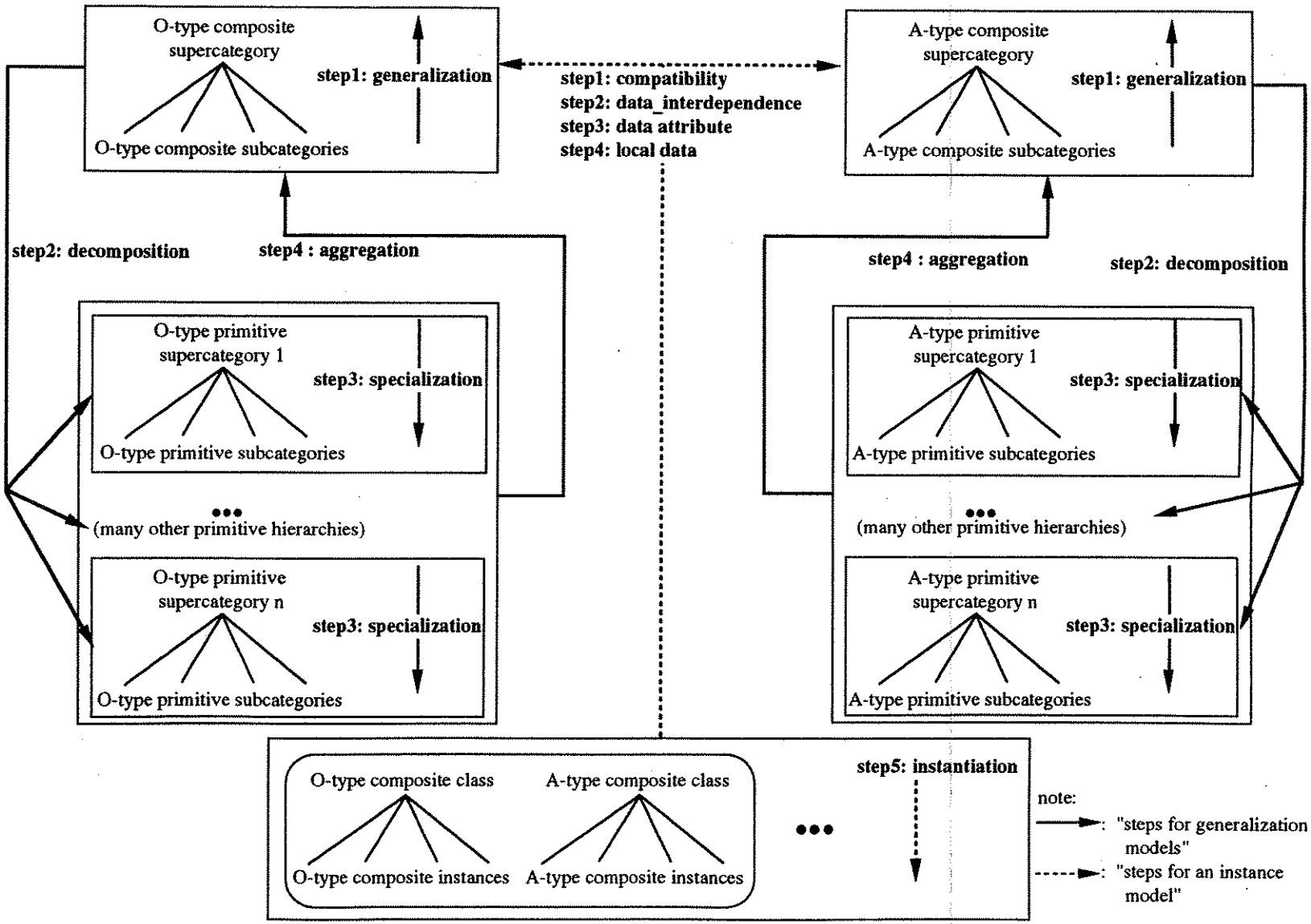


Figure 5.22. Summary of procedures for developing generalization and instance models.

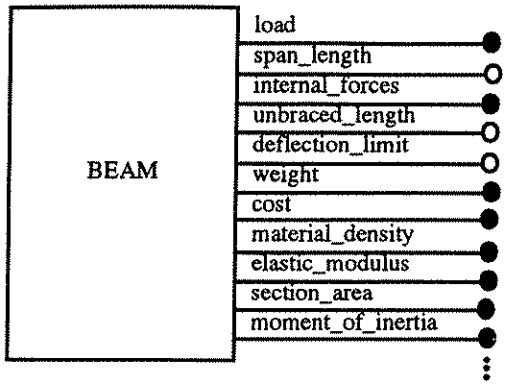


Figure 6.1. BEAM category.

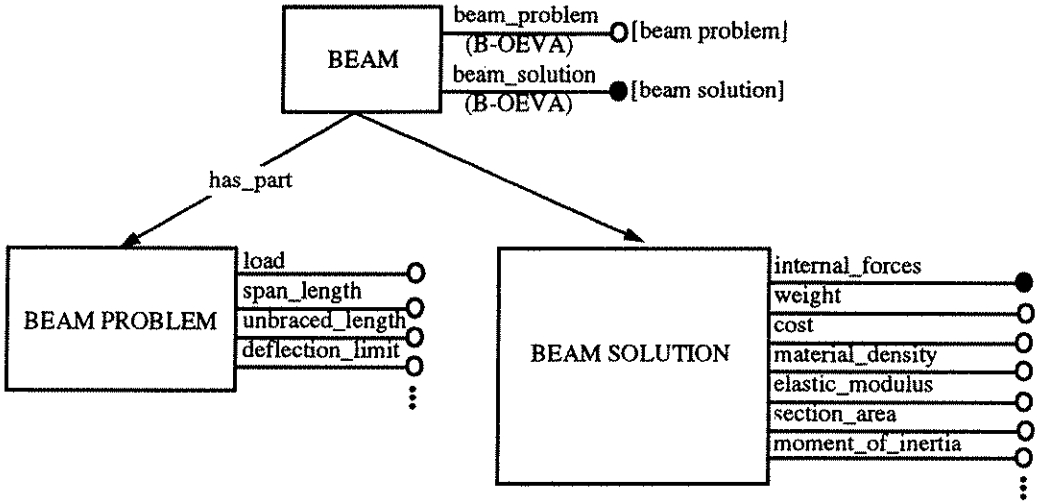


Figure 6.2. Decomposition of the BEAM category.

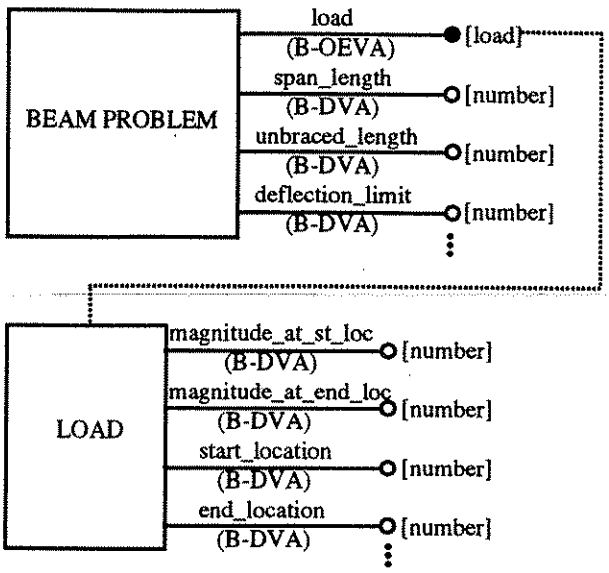


Figure 6.3. BEAM PROBLEM entity category with range categories.

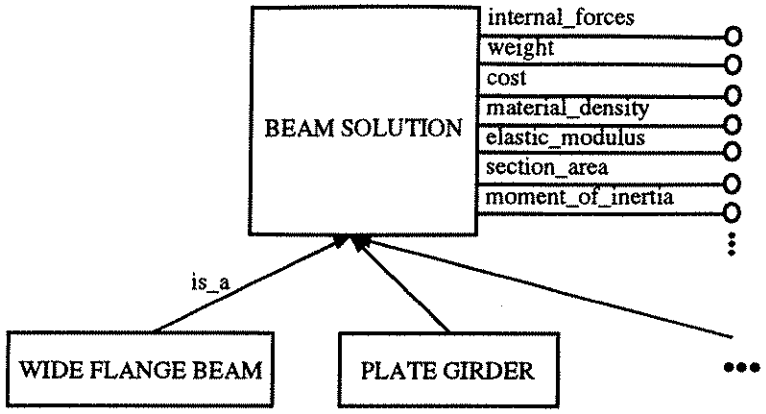


Figure 6.4. Composite generalization hierarchy with O-type entities for beam design alternatives.

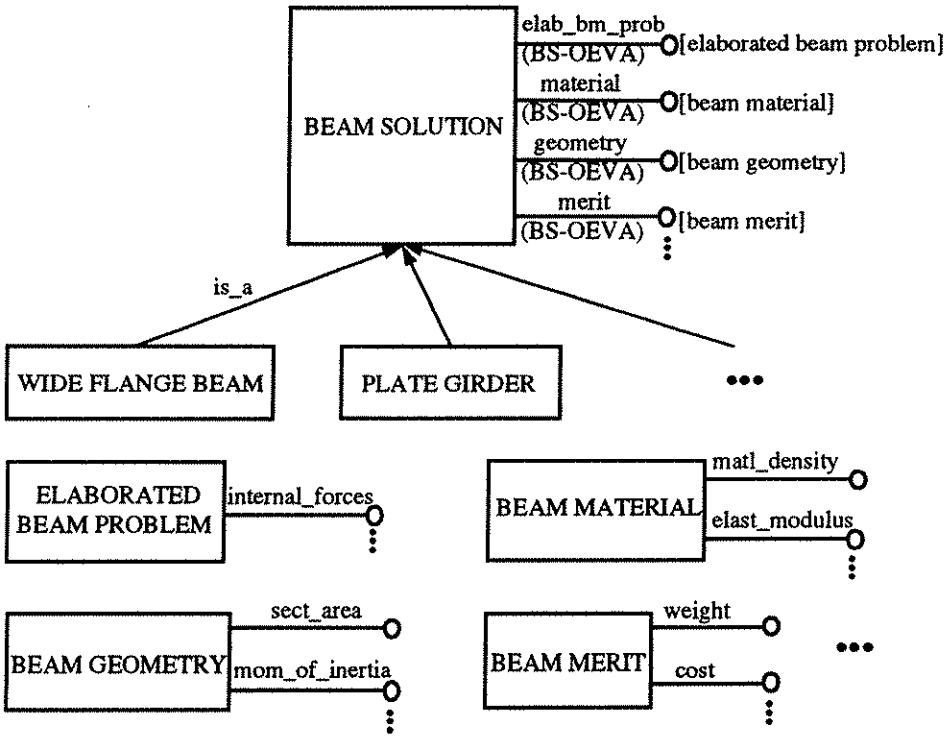
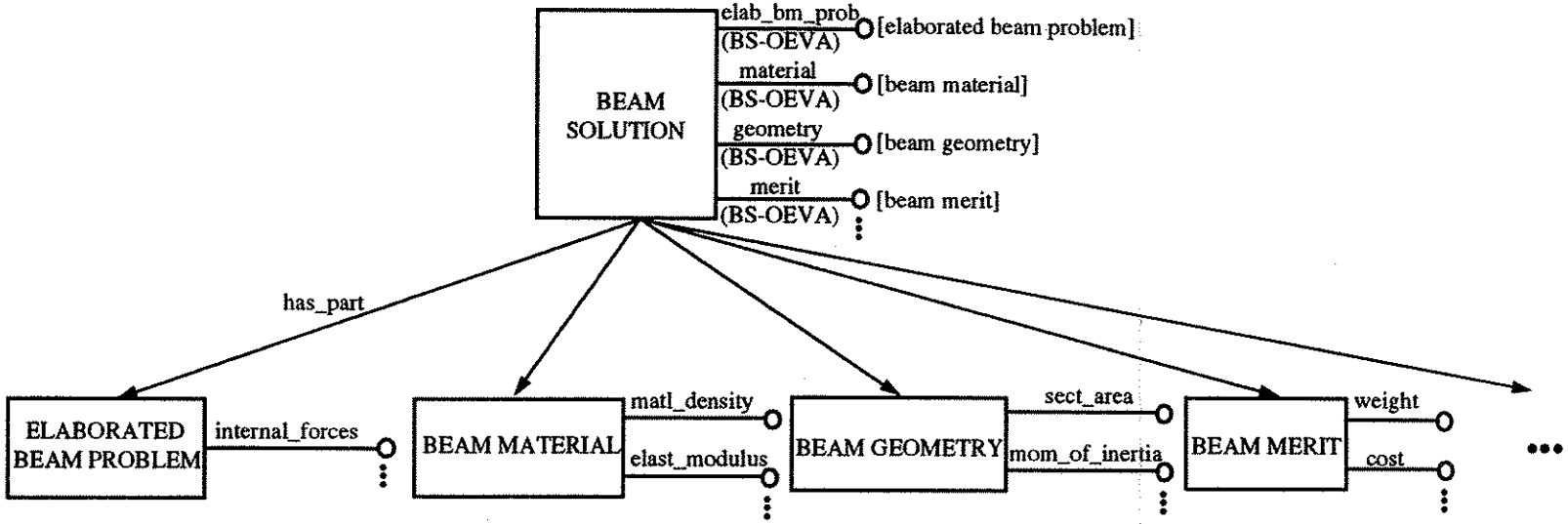
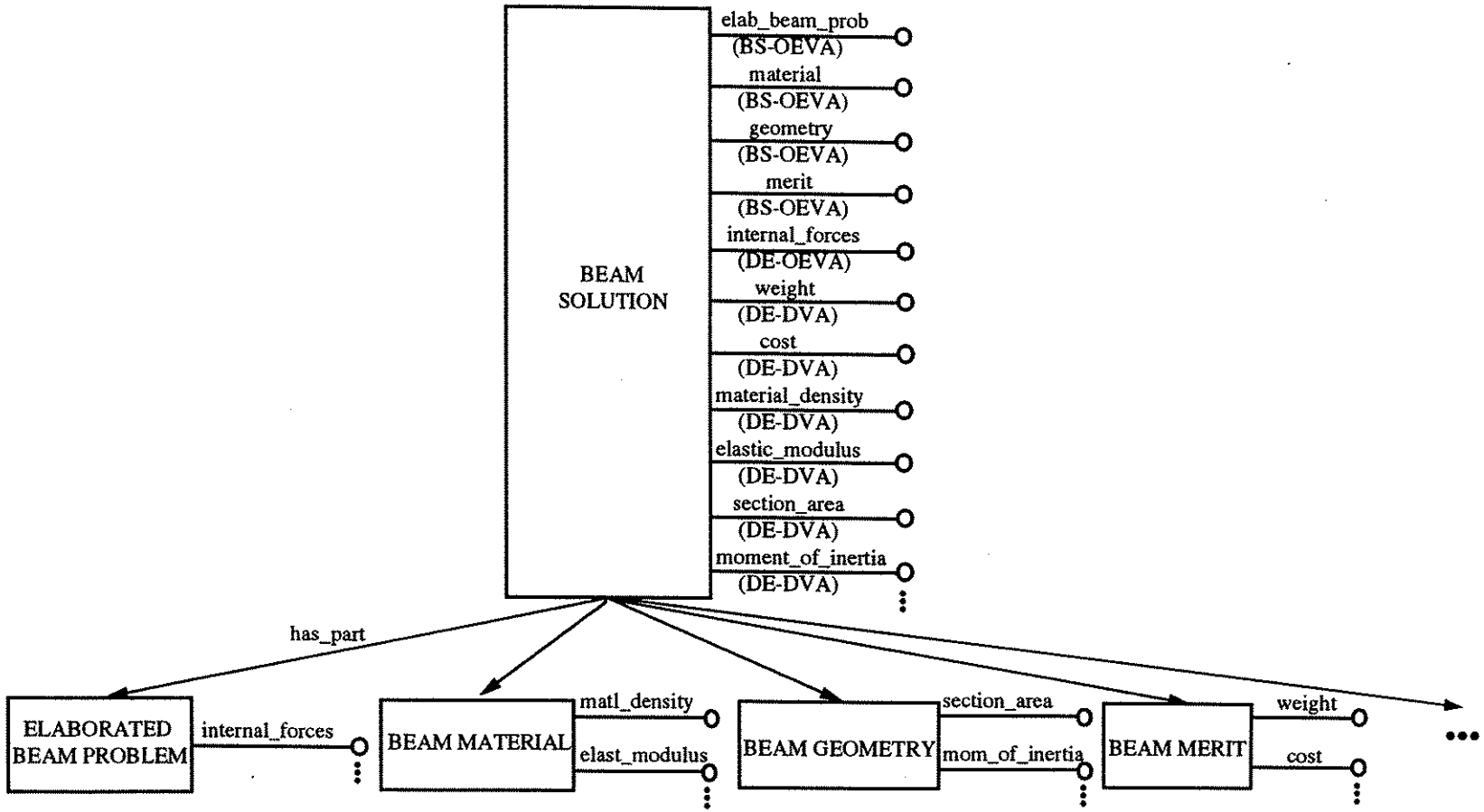


Figure 6.5. Modified composite generalization hierarchy for beam design alternatives.



a. Decomposition of the BEAM SOLUTION category.

Figure 6.6. Decomposition of the BEAM SOLUTION category.



b. An expanded representation of the BEAM SOLUTION category.

Figure 6.6. Decomposition of the BEAM category (continued).

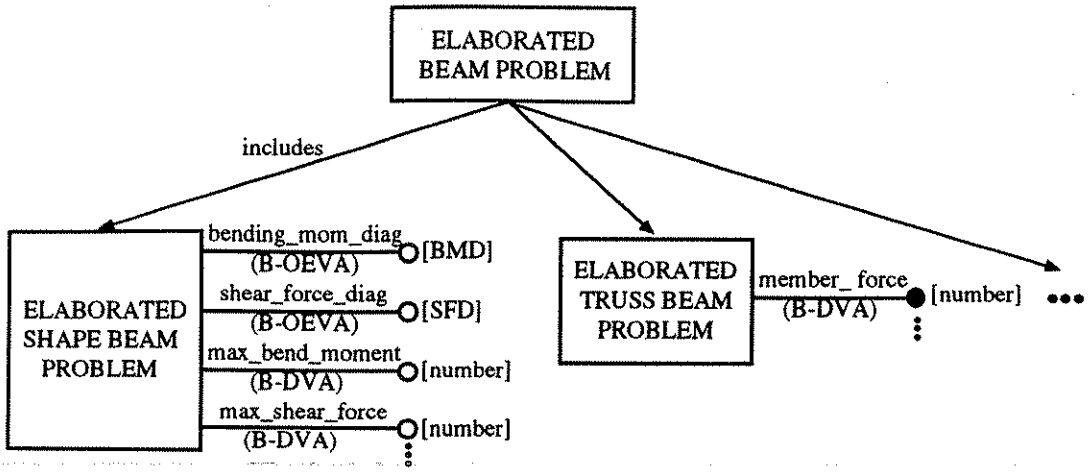
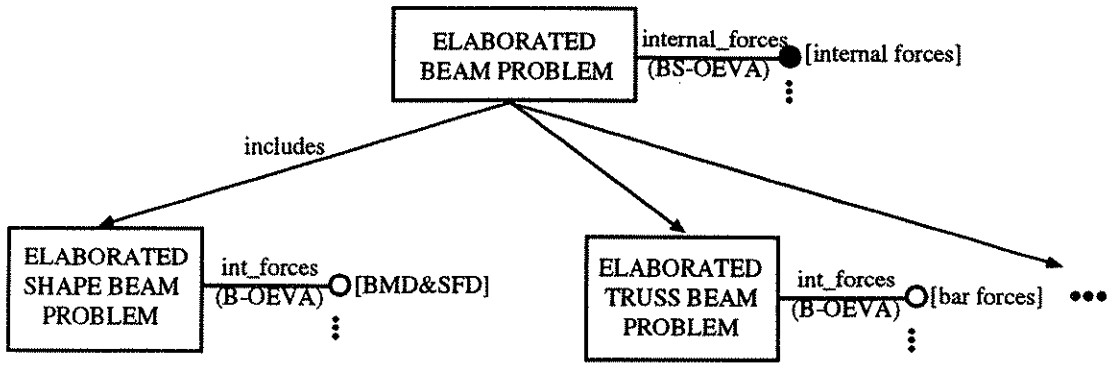
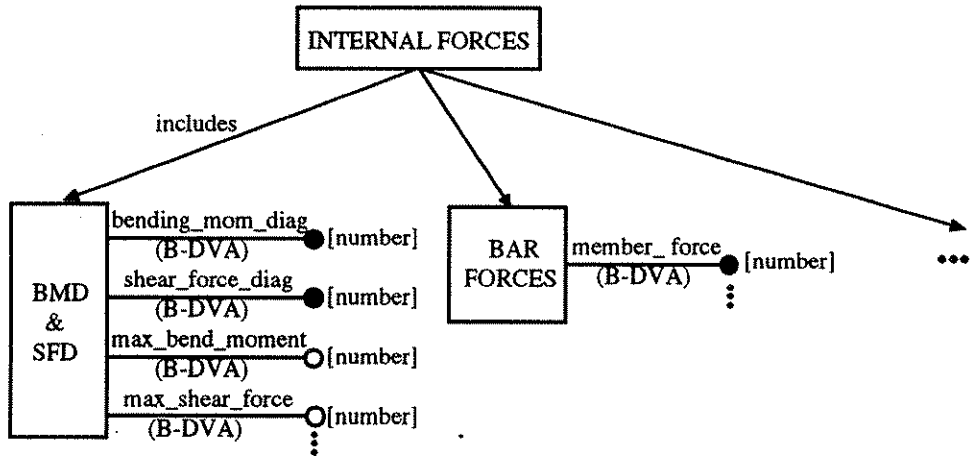


Figure 6.7. Primitive hierarchy for the elaborated beam problem OEVA.



a. Primitive hierarchy.



b. Generalization hierarchy for range categories of OEVAs in primitive hierarchy.

Figure 6.8. A different primitive hierarchy
for the elaborated beam problem OEVA.

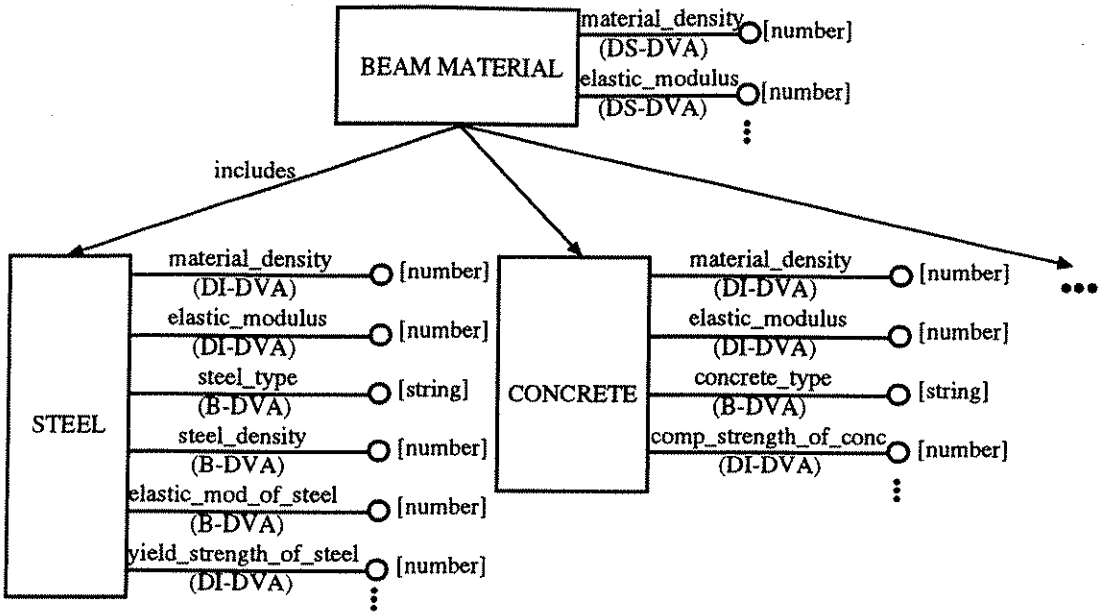


Figure 6.9. Primitive hierarchy for the beam material OEVA.

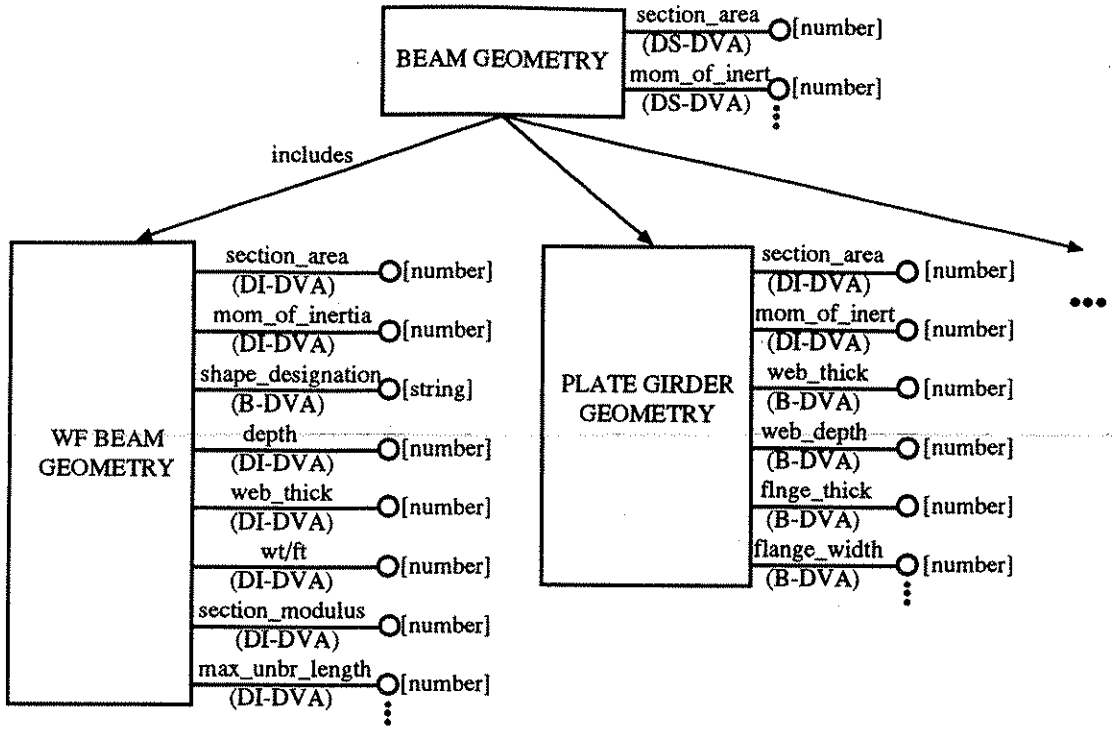


Figure 6.10. Primitive hierarchy for the beam geometry OEVA.

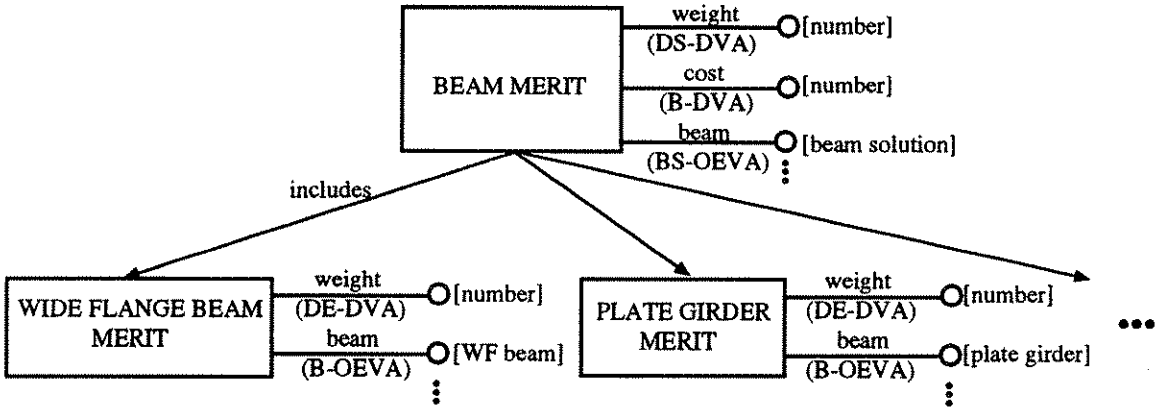


Figure 6.11. Primitive hierarchy for the beam merit OEVA.

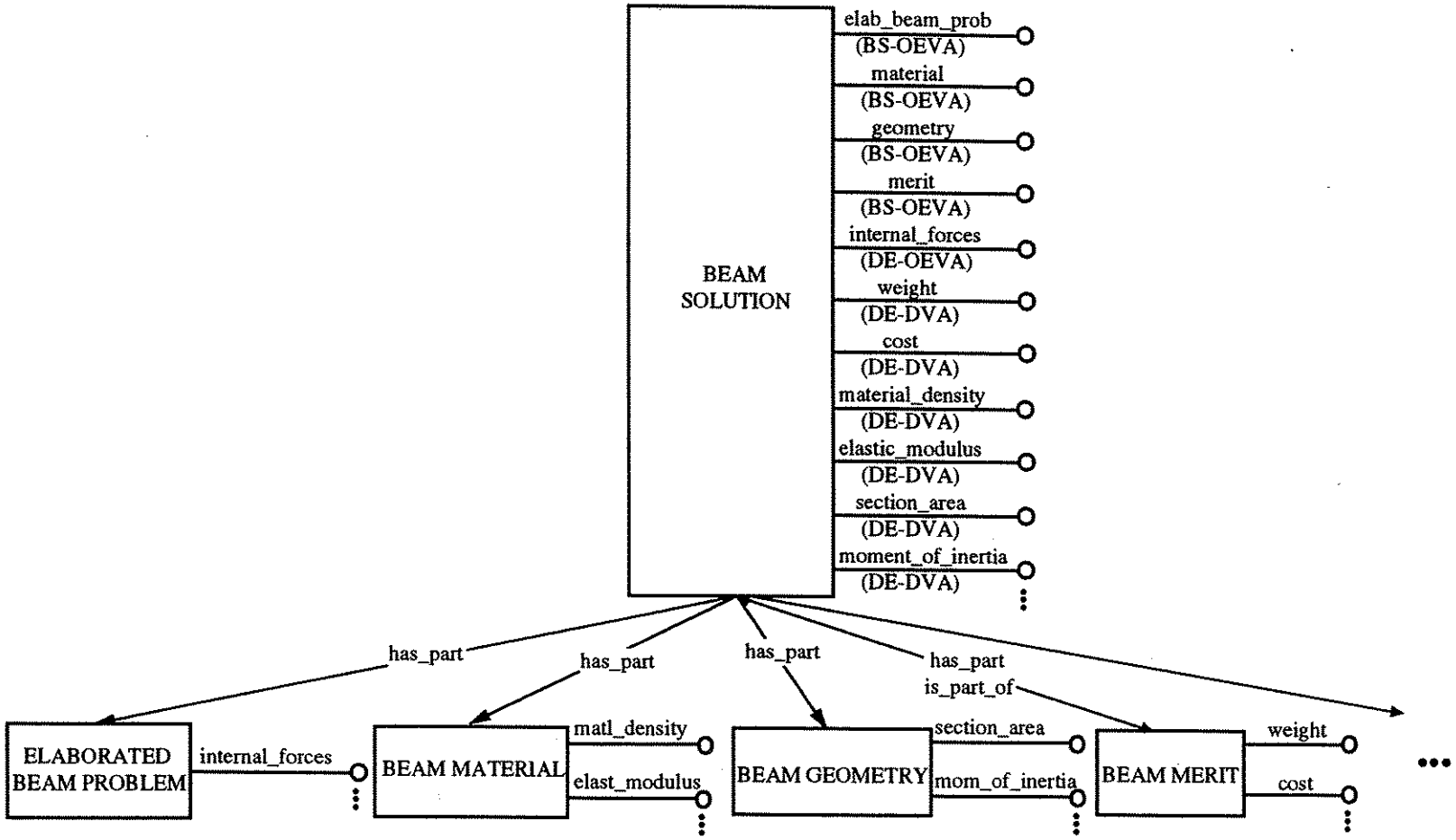


Figure 6.12. Two way relationship between the BEAM SOLUTION category and the BEAM MERIT category.

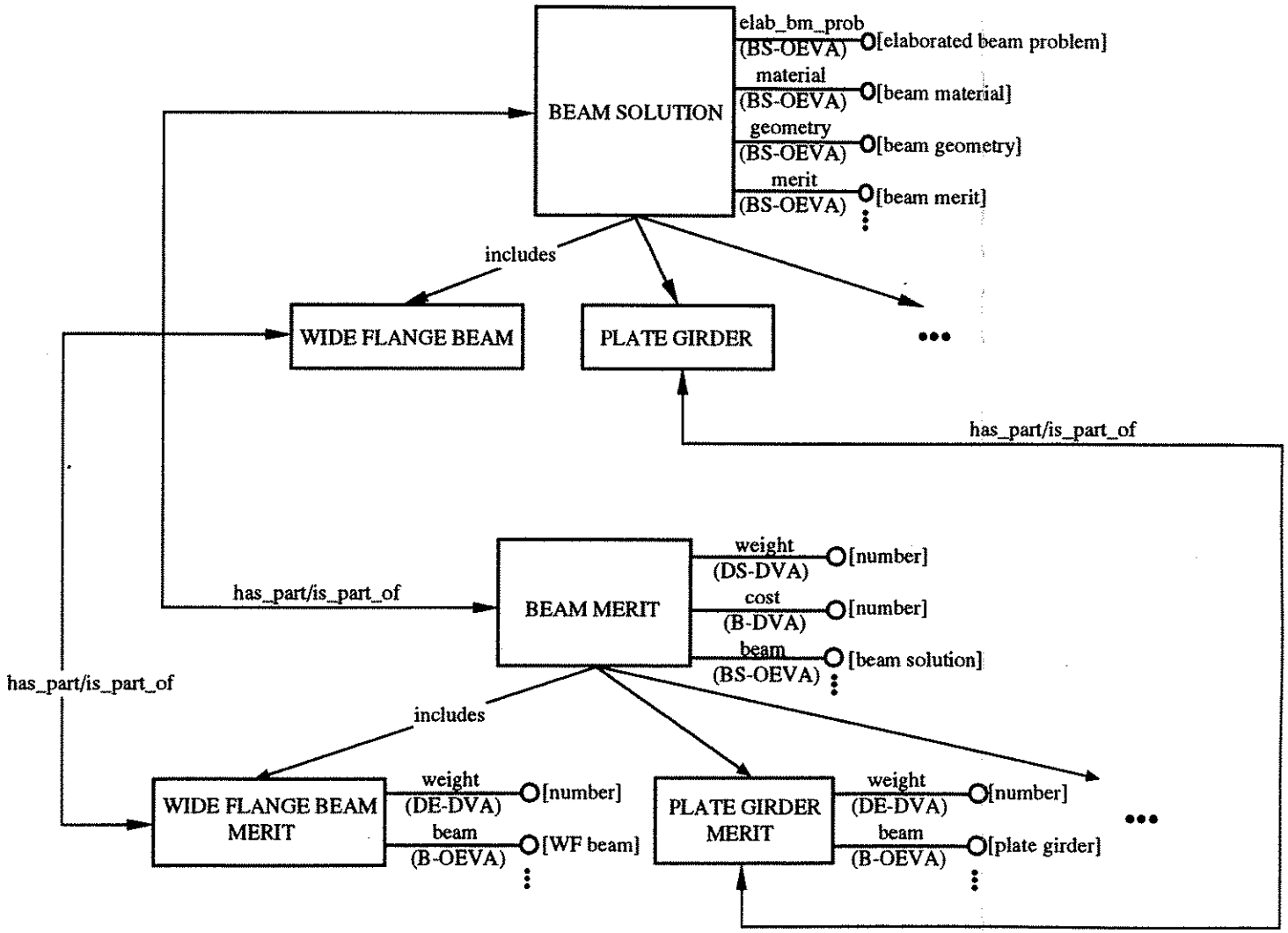


Figure 6.13. Two way relationship between categories of the beam merit primitive hierarchy and categories of the O-type composite hierarchy for beam design alternatives.

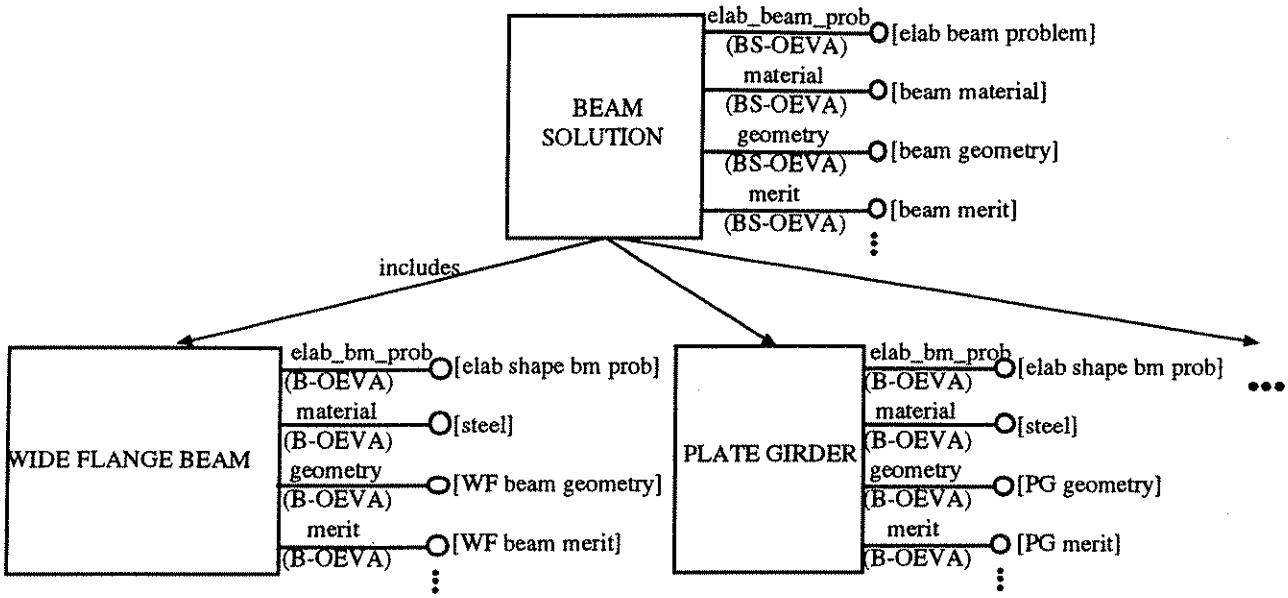


Figure 6.14. Value set specialization for O-type composite generalization hierarchy for beam design alternatives.

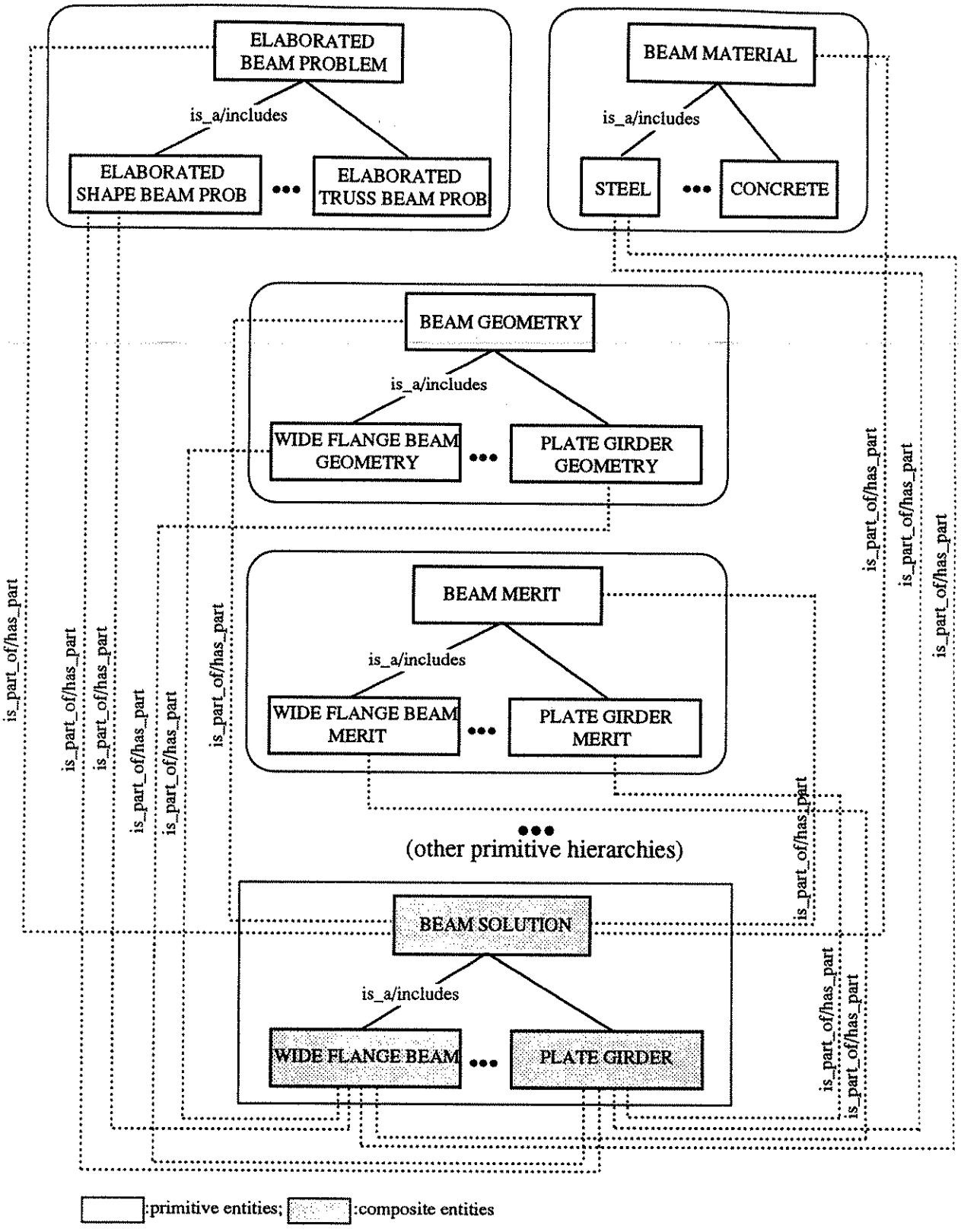


Figure 6.15. Relationships between primitive and composite generalization hierarchies for the product model for beam design alternatives.

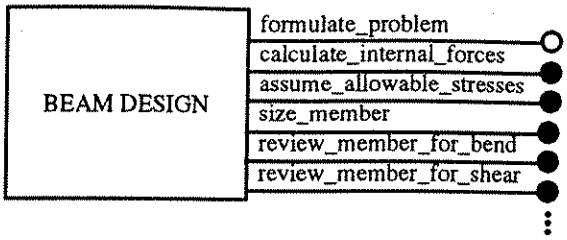


Figure 6.16. BEAM DESIGN category.

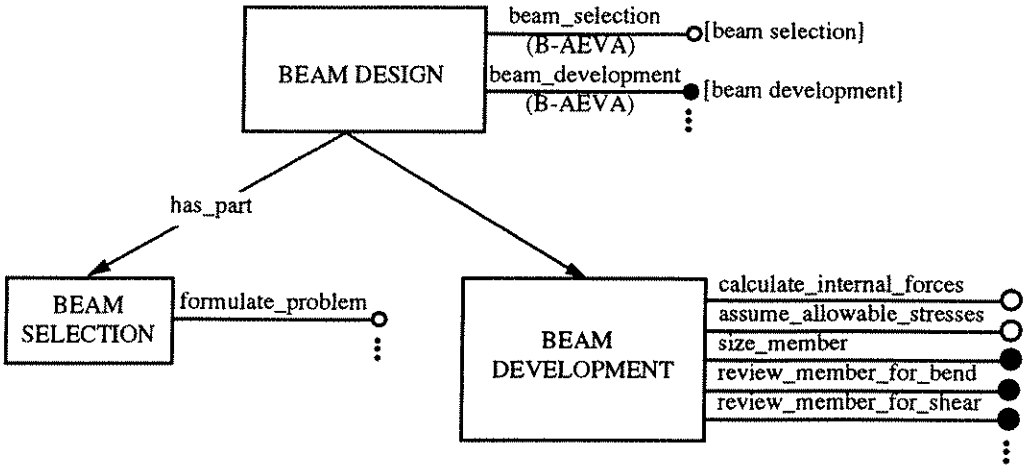


Figure 6.17. Decomposition of BEAM DESIGN category.

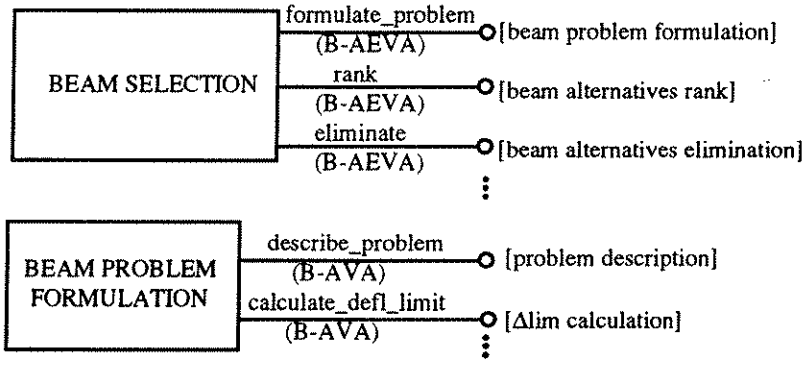


Figure 6.18. BEAM SELECTION category with its range category.

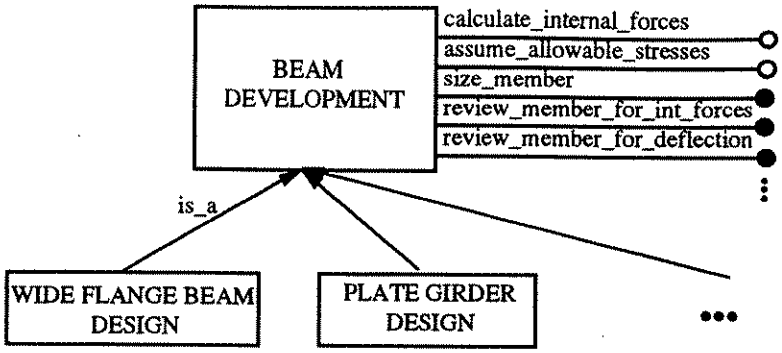


Figure 6.19. Composite generalization hierarchy with A-type entities for beam design.

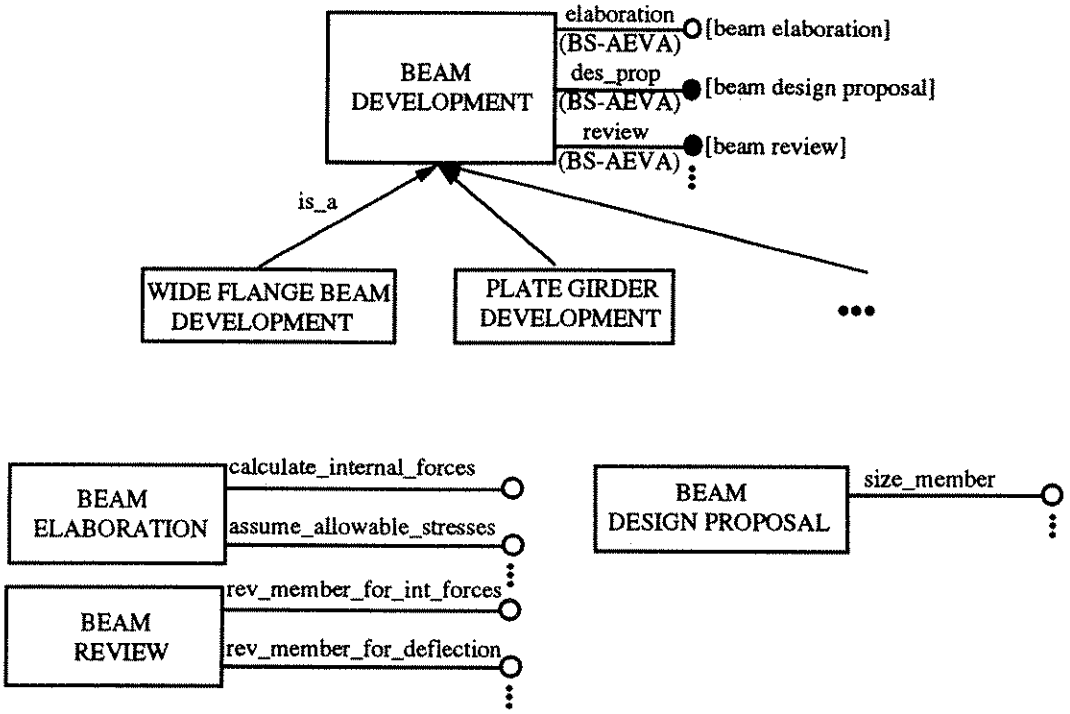


Figure 6.20. Modified composite generalization hierarchy with A-type entities for beam design.

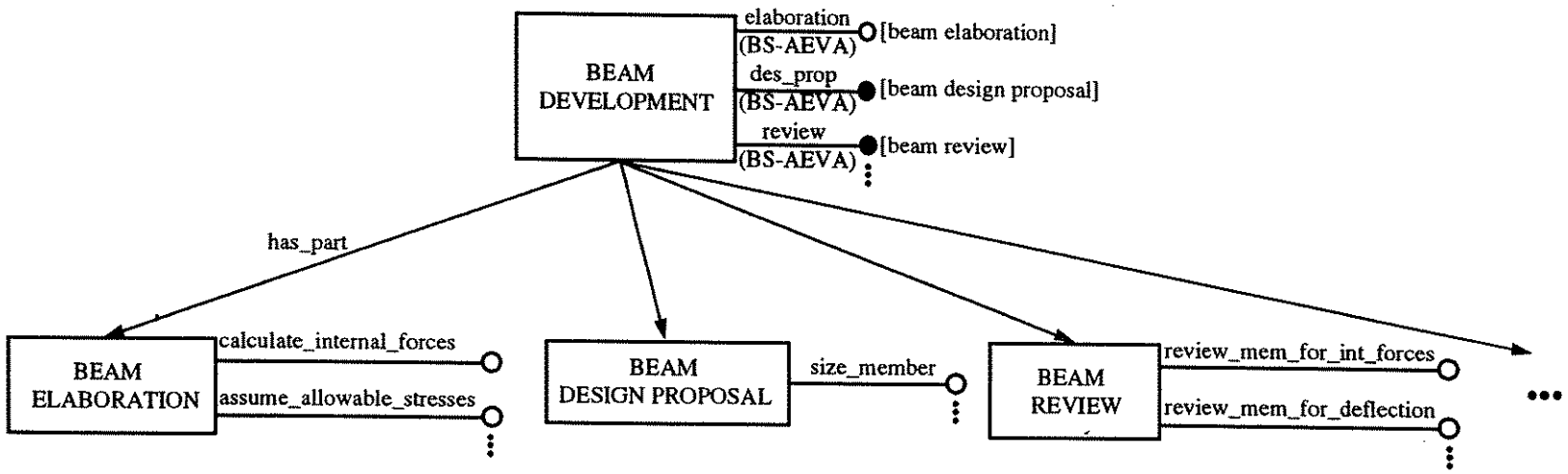
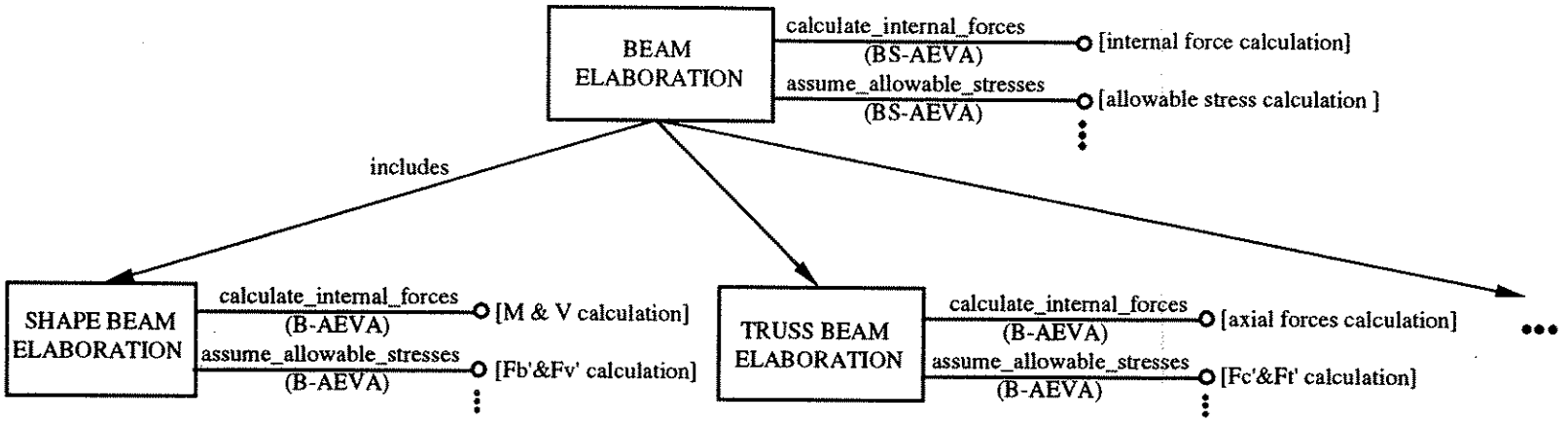
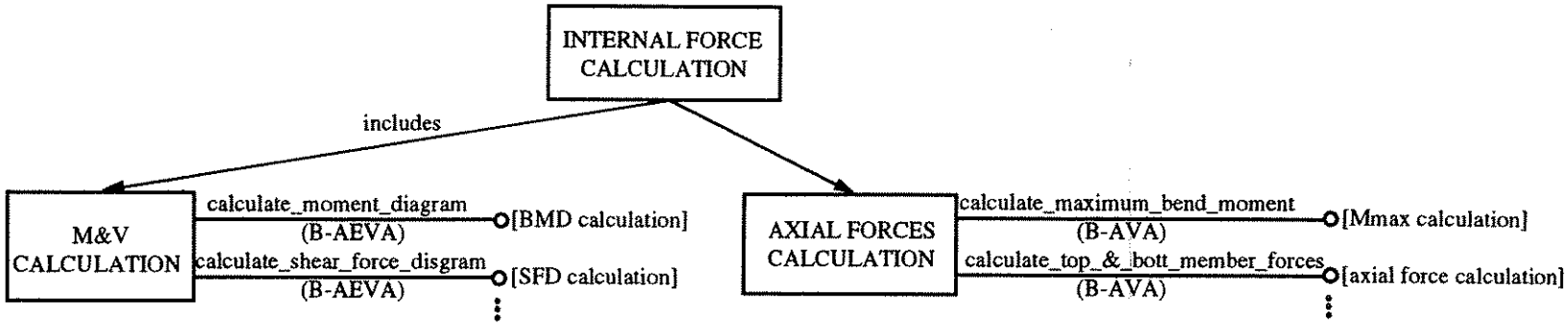


Figure 6.21. Decomposition of the BEAM DEVELOPMENT category.

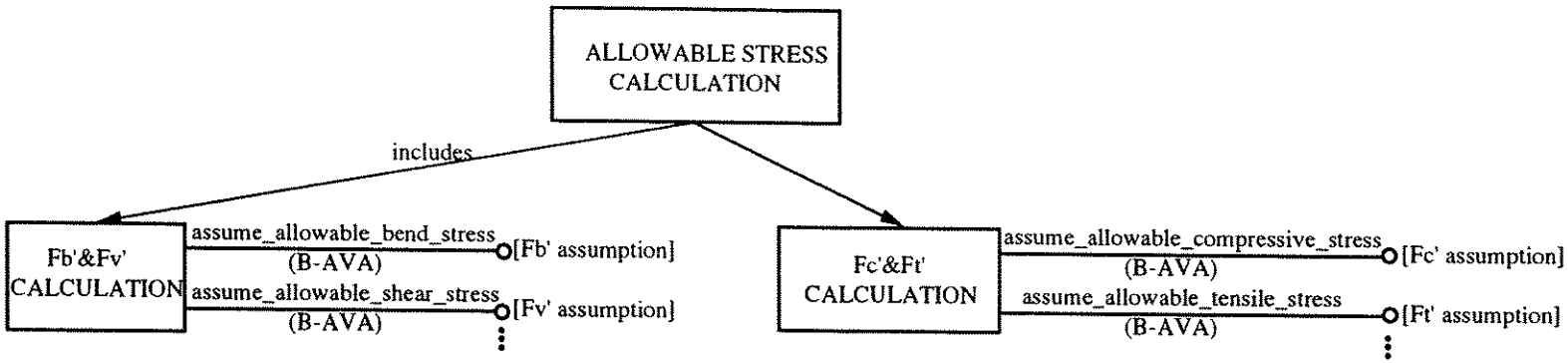


a. Primitive hierarchy .

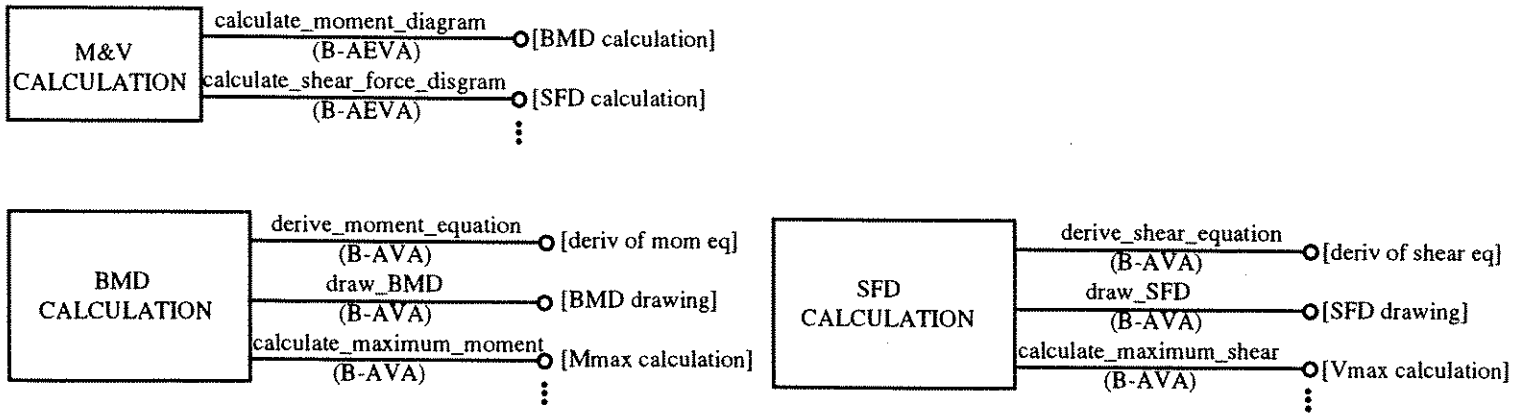


b. Generalization hierarchies for range categories of calculate_internal_forces AEVAs in primitive hierarchy.

Figure 6.22. Primitive hierarchy for the elaboration AEVA.



c. Generalization hierarchies for range categories of assume_allowable_stresses AEVAs in primitive hierarchy .



d. M&V CALCULATION category with range categories.

Figure 6.22. Primitive hierarchy for the elaboration AEVA (continued).

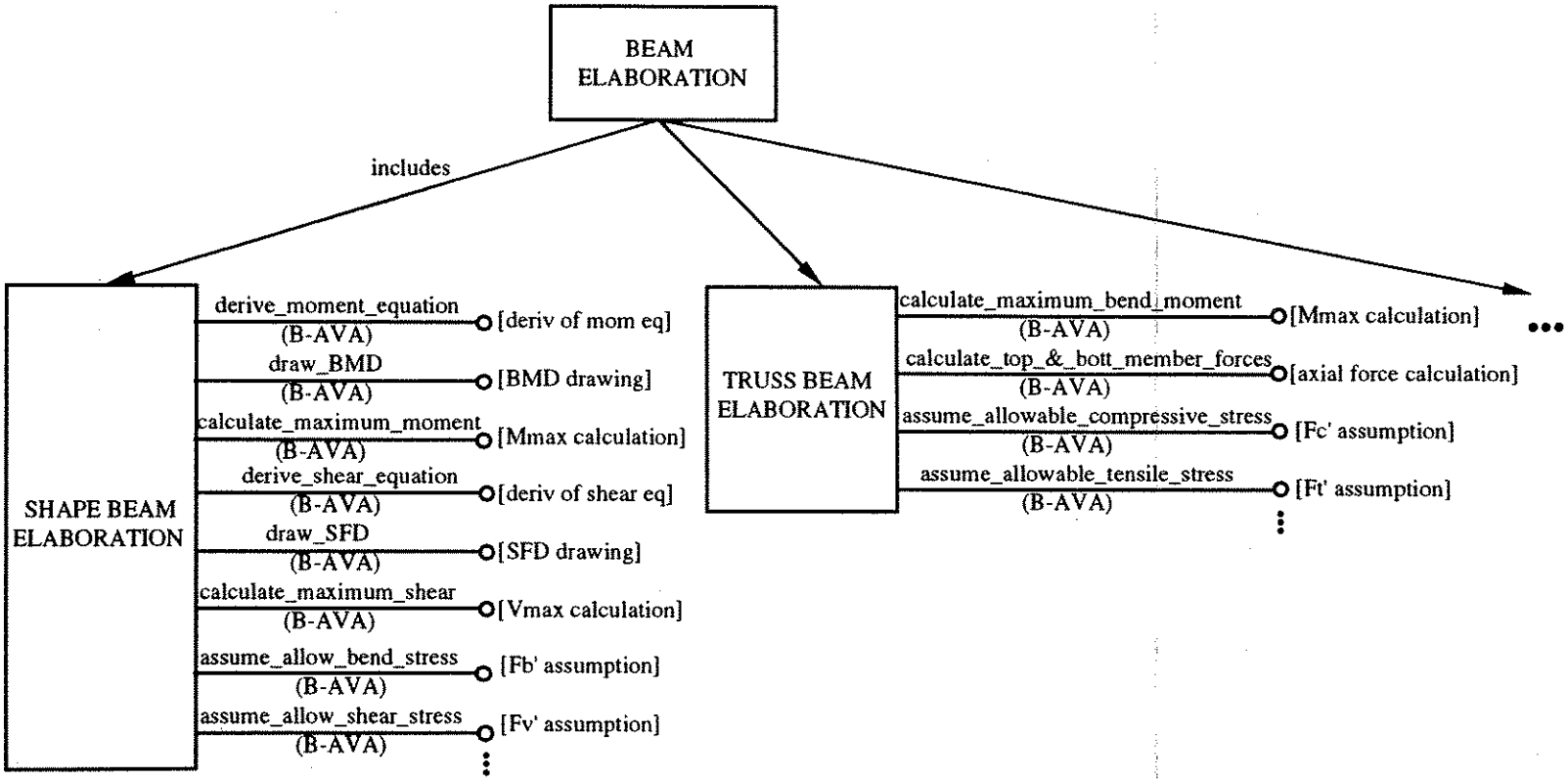
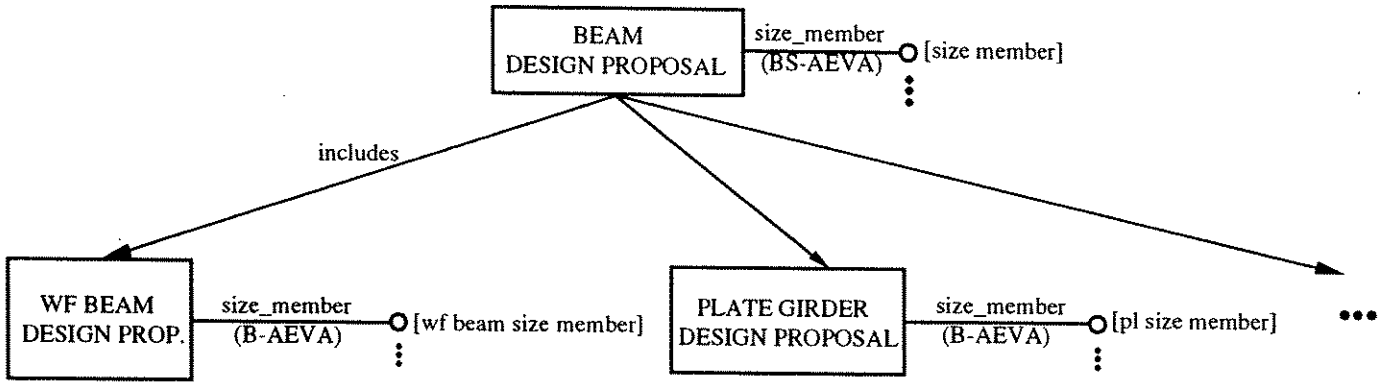
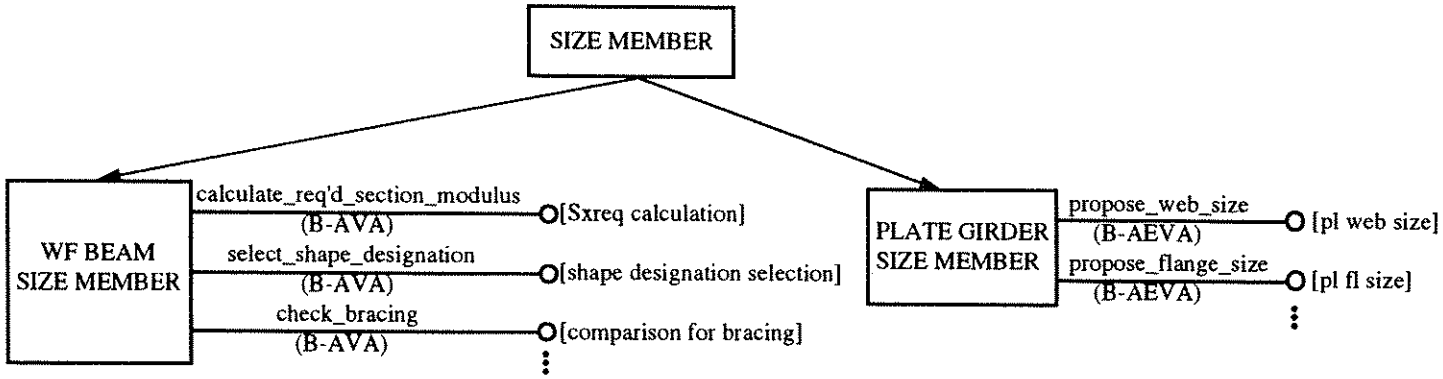


Figure 6.23. A different primitive hierarchy for the elaboration AEVA.

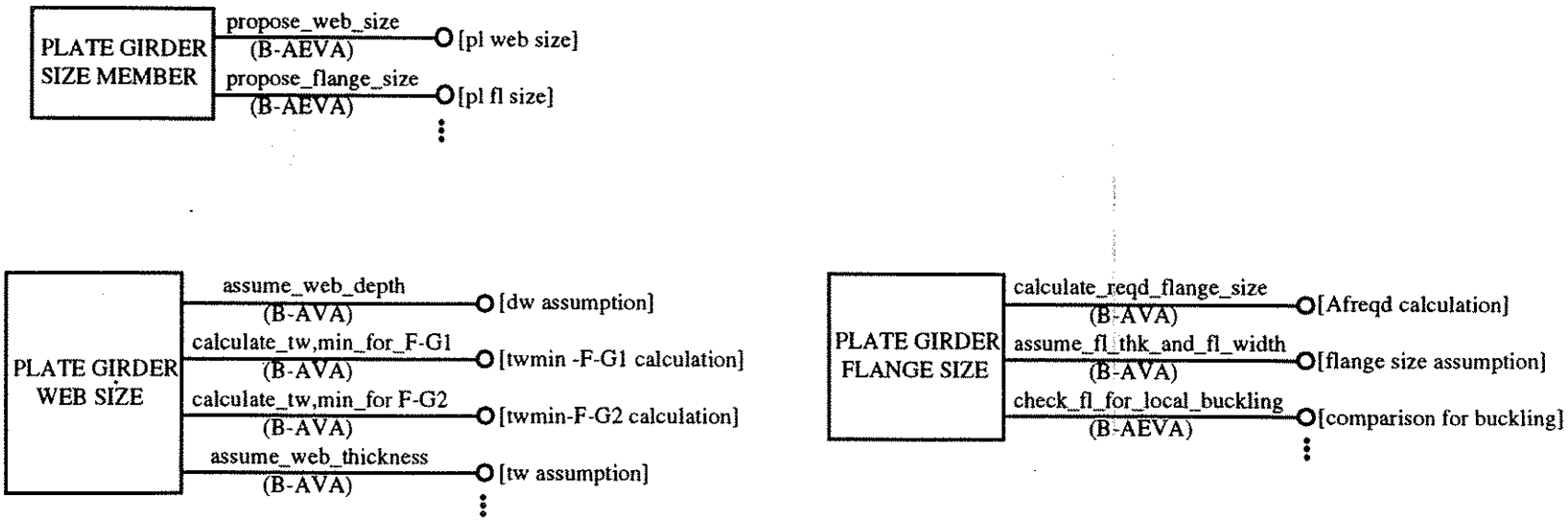


a. Primitive hierarchy.



b. Generalization hierarchy for range categories of AEVAs in primitive hierarchy.

Figure 6.24. Primitive hierarchy for the design proposal AEVA.



c. PLATE GIRDER SIZE MEMBER category with range categories.

Figure 6.24. Primitive hierarchy for the design proposal AEVA (continued).

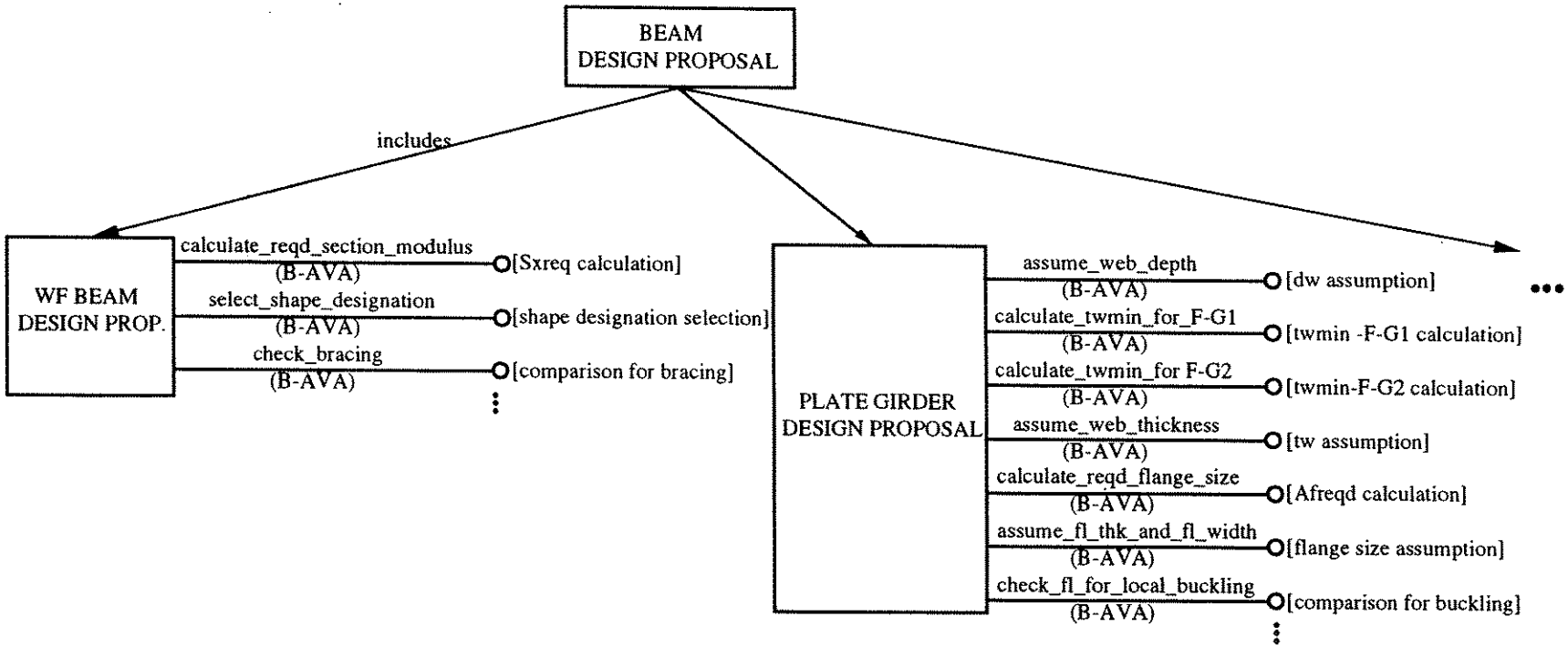
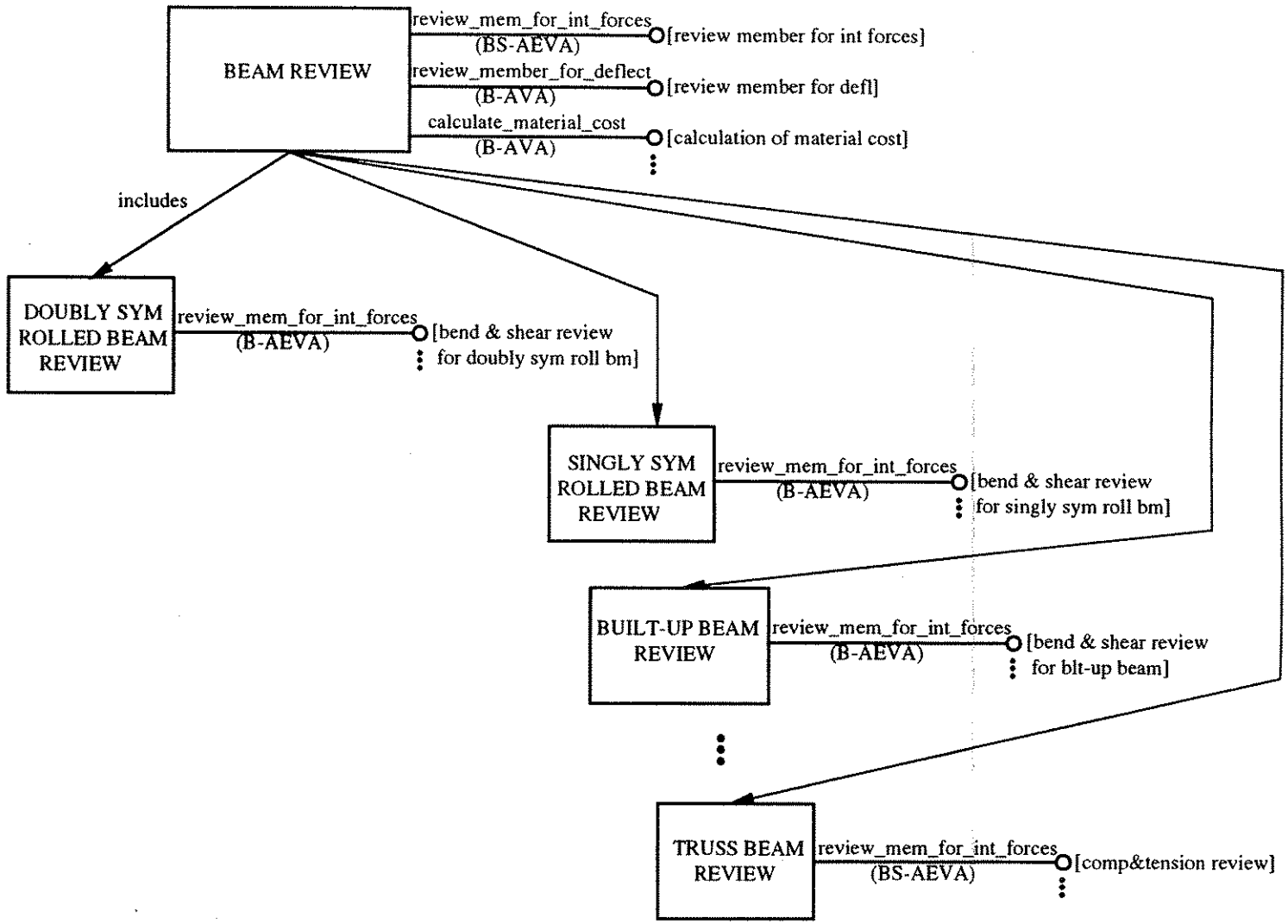
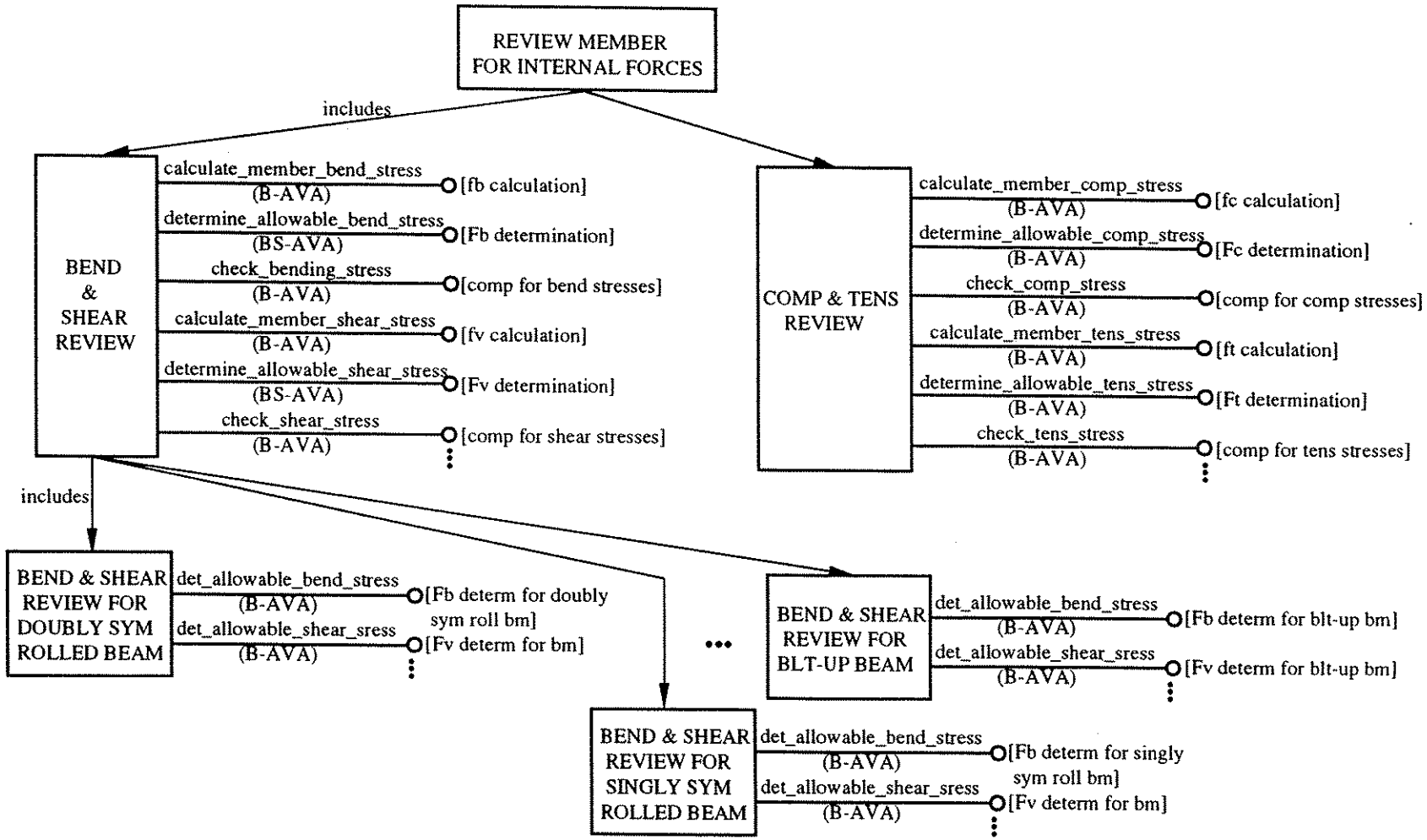


Figure 6.25. A different representation of primitive hierarchy for the design proposal AEVA.



a. Primitive hierarchy.

Figure 6.26. Primitive hierarchy for the review AEVA.



b. Generalization hierarchy for range categories of AEVAs in primitive hierarchy.

Figure 6.26. Primitive hierarchy for the review AEVA (continued).

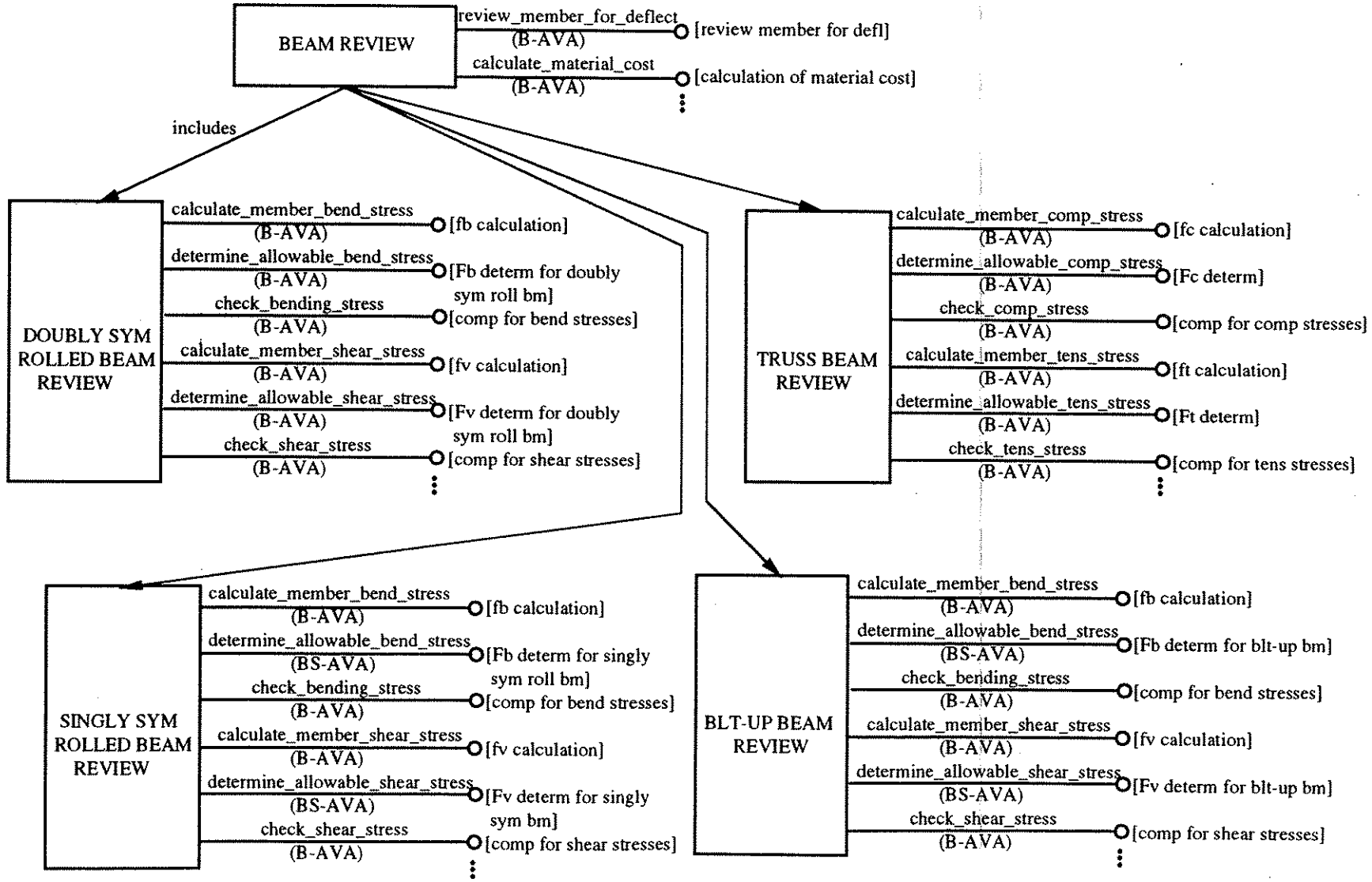


Figure 6.27. A different primitive hierarchy for the review AEVA.

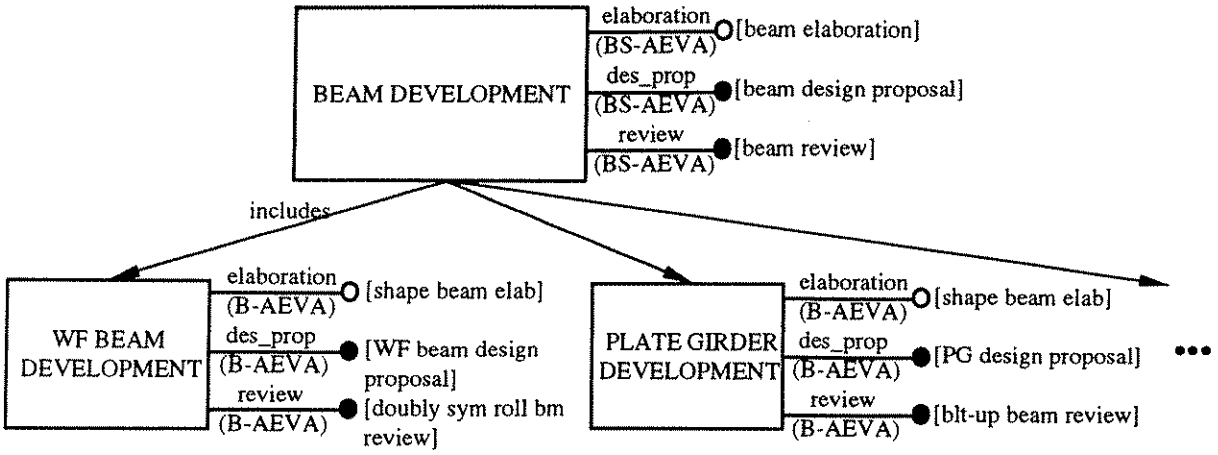


Figure 6.28. Value set specialization for A-type composite generalization hierarchy for beam design.

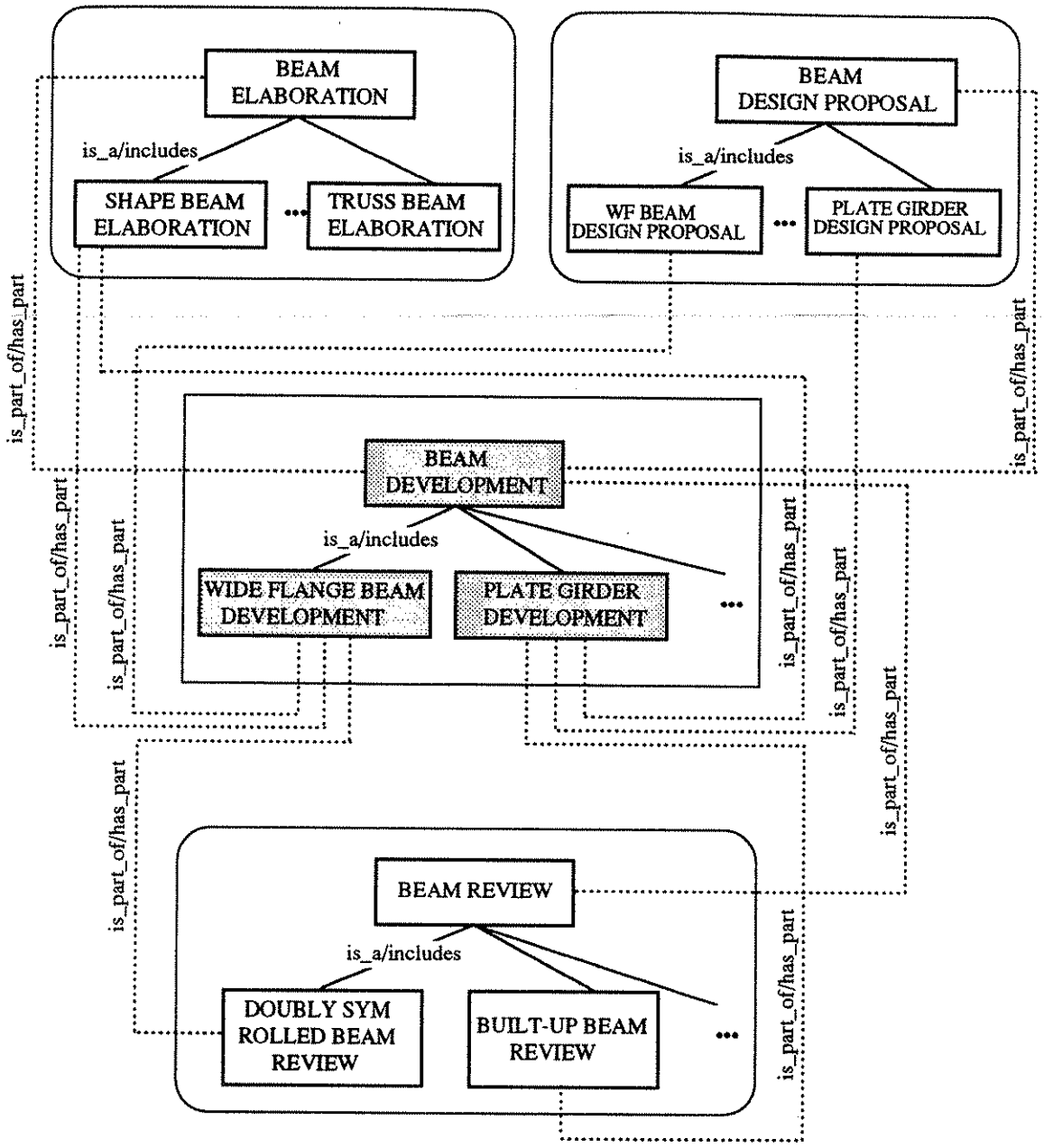
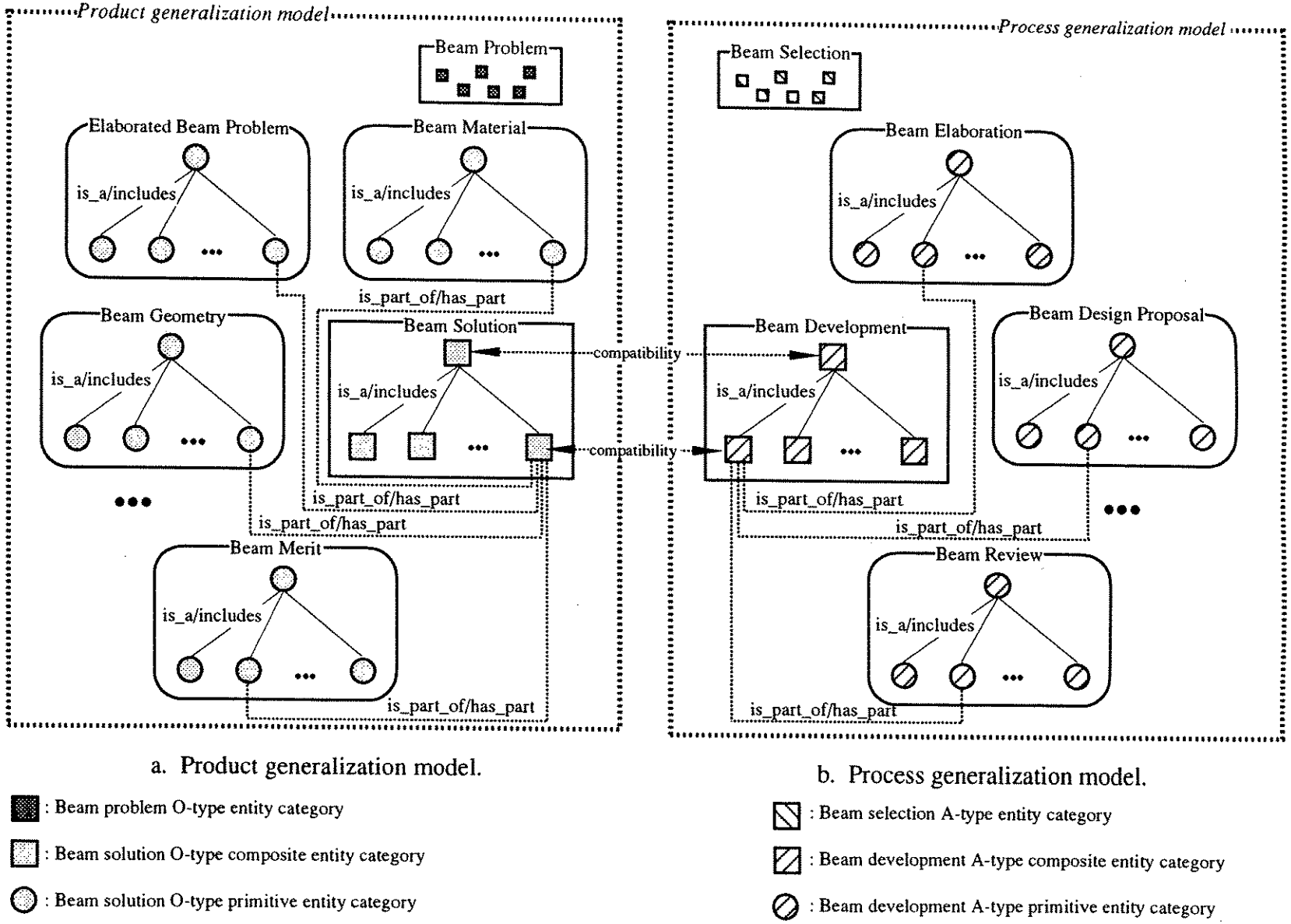


Figure 6.29. Relationships between primitive and composite generalization hierarchies for the process model for beam design.



a. Product generalization model.

b. Process generalization model.

: Beam problem O-type entity category

: Beam solution O-type composite entity category

: Beam solution O-type primitive entity category

: Beam selection A-type entity category

: Beam development A-type composite entity category

: Beam development A-type primitive entity category

Figure 6.30. Compatibility between O-type and A-type composite categories.

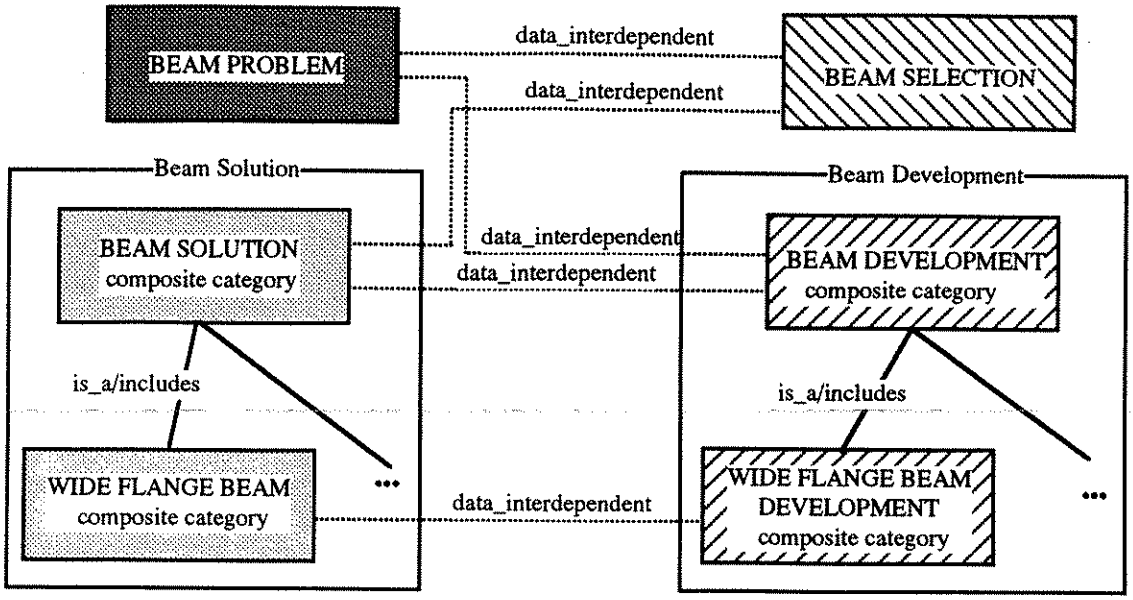


Figure 6.31. Data_interdependent relationships between composite categories.

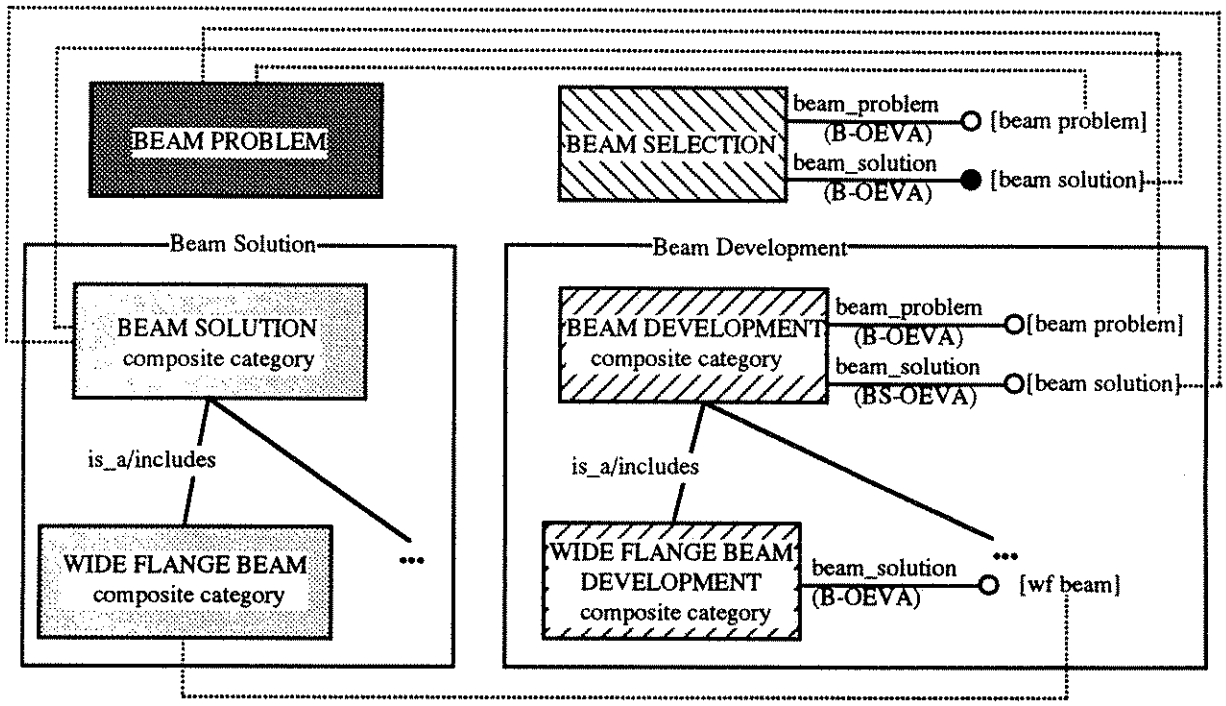
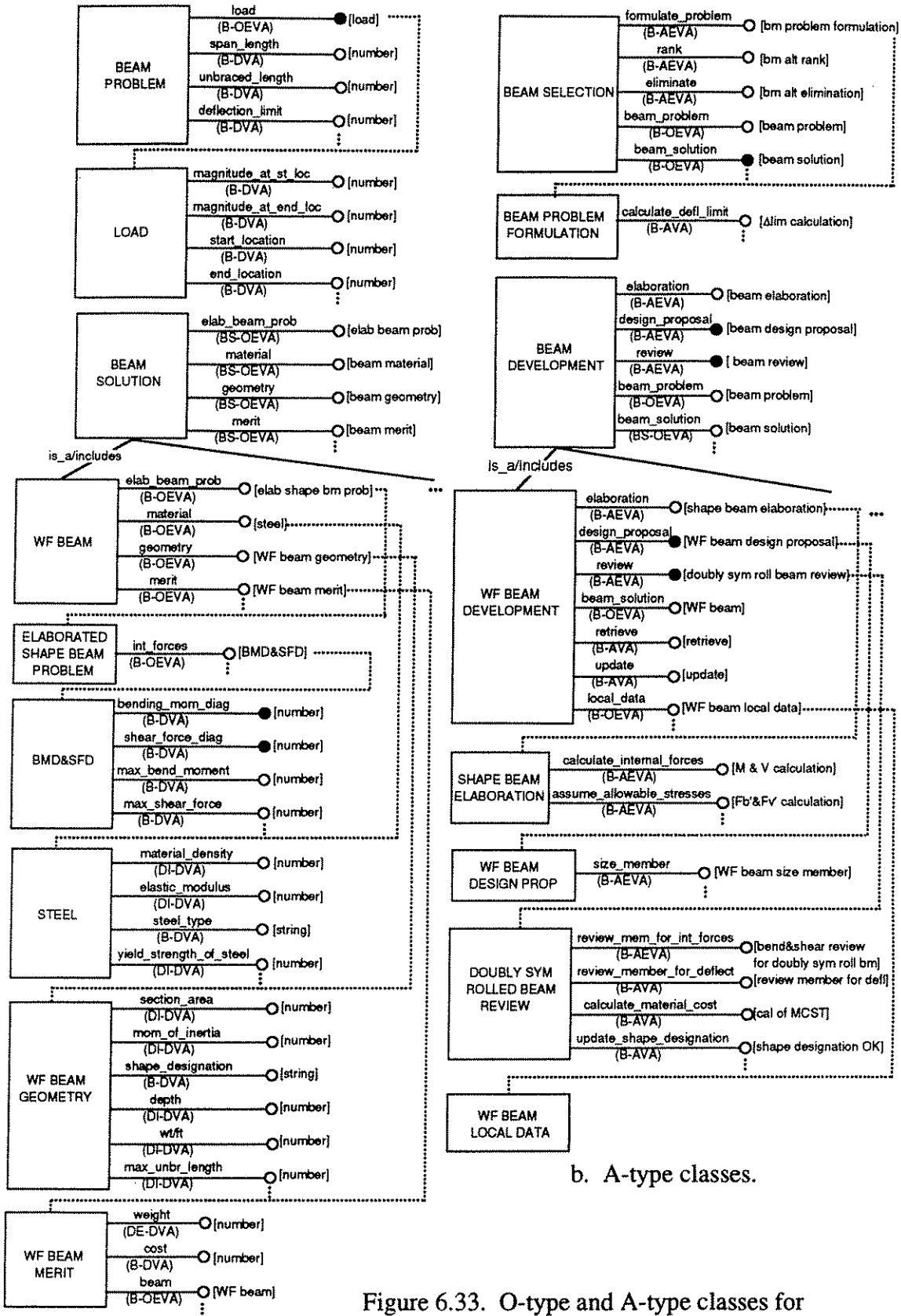


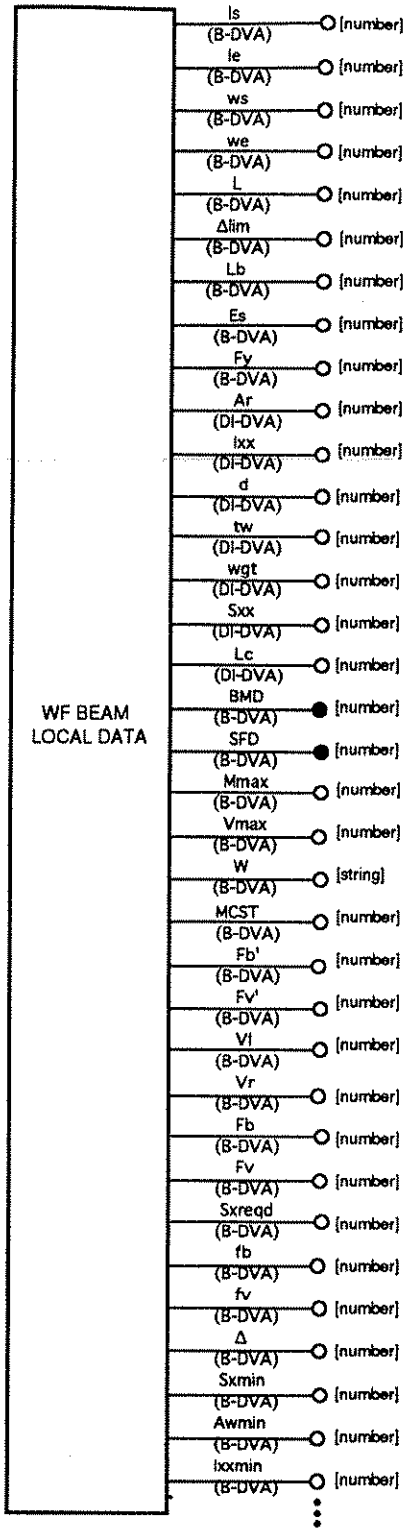
Figure 6.32. Formalization of data_interdependent relationships as data attributes.



a. O-type classes.

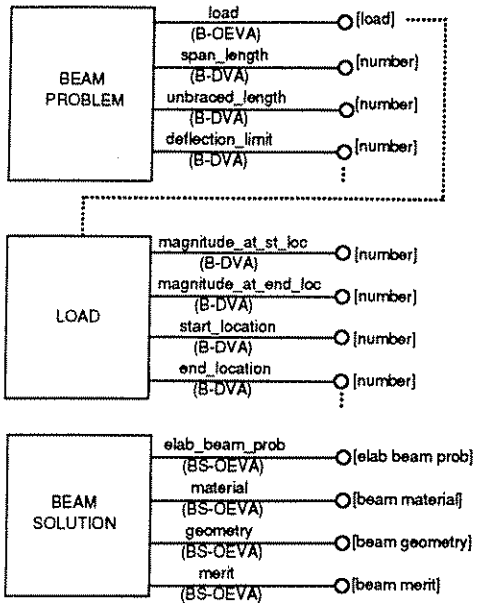
b. A-type classes.

Figure 6.33. O-type and A-type classes for wide flange beam design.

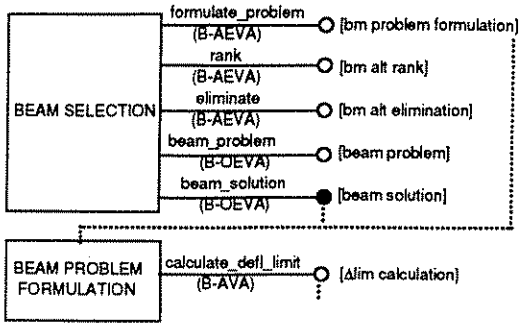


c. WF BEAM LOCAL DATA class.

Figure 6.33. O-type and A-type classes for wide flange beam design (continued).



a. O-type classes.



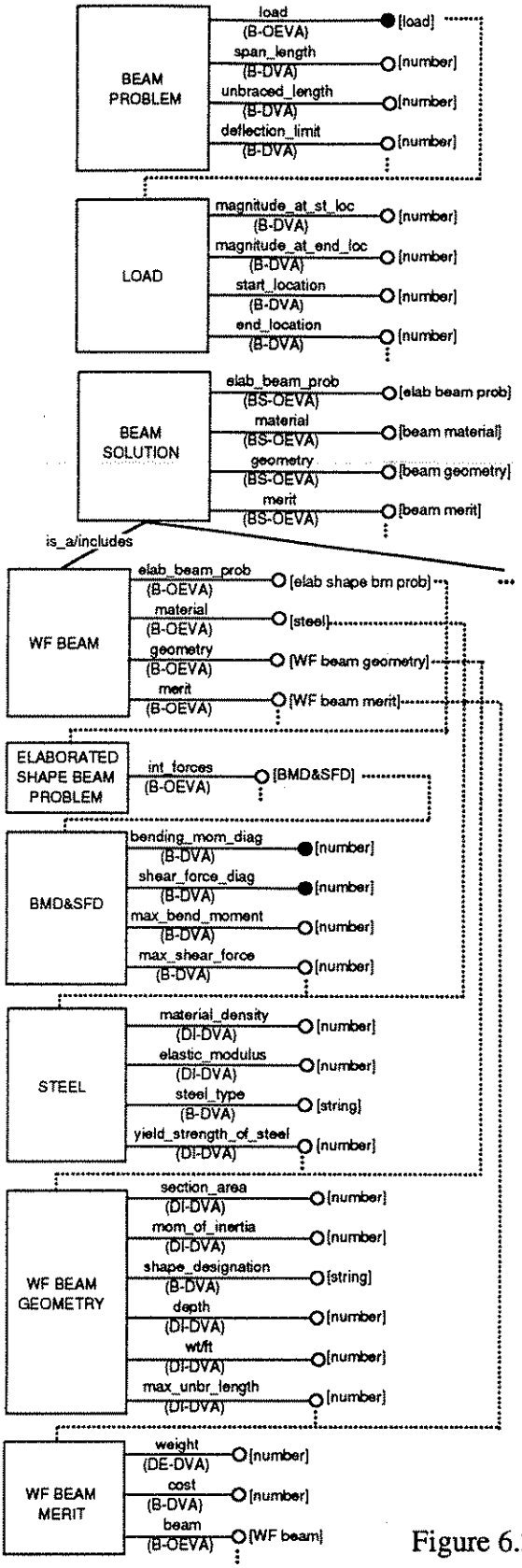
c. A-type classes.

action(Δlim calculation)
input(L:number)
output(Δlim:number)
expressions

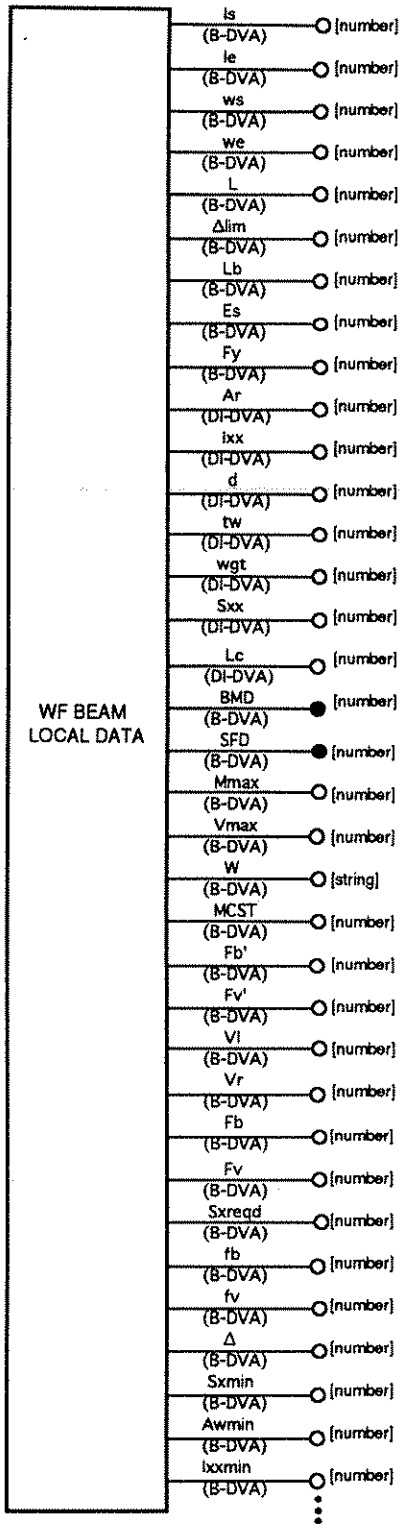
L=beam_problem.span_length
Δlim=L/360
beam_problem.deflection_limit=Δlim

b. Actions.

Figure 6.34. O-type and A-type classes with actions for wide flange beam selection tasks.



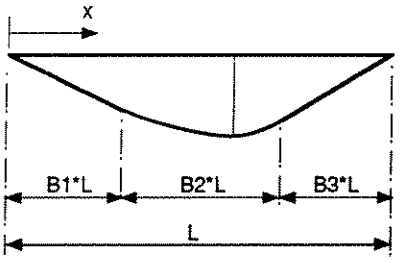
a. O-type classes.



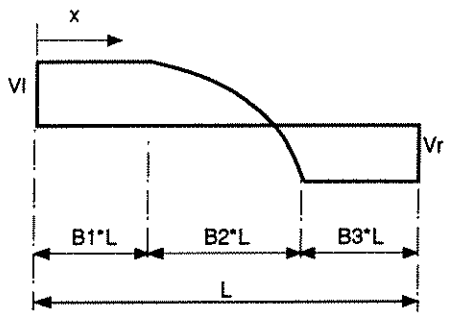
b. WF BEAM LOCAL DATA class.

Figure 6.35. O-type and A-type classes for wide flange beam elaboration tasks.

```
action(initial calculation)
input(L,ls,le,we,ws:number)
output(B1,B2,B3,A,Vl ,Vr:number)
expressions

B1 = ls/L
B2 = (le-ls)/L
B3 = 1-le/L
A = we/ws
Vl = ws*B2*L((2+A)*B2+3*(A+1)*B3)/6
Vr = ws*B2*L(3*(A+1)*(1-B3)-(2+A)*B2)/6

action(derivation of moment equation)
input(L,ws, we, Vl,Vr,A:number)
output(BMD(x):number)
expressions

Δx = L/20
iter i = 1,21
[x = Δx*(i-1)
(x<ls)
ifTrue[M(x)=Vl*x; BMD(x)=M(x)]
ifFalse[(ls =< x <le)
        ifTrue[M(x) = V*x-ws*(x-B1*L)*(x+B1*L)/2
       -ws*(A-1)*((x-B1*L)**2)*(x+2*B1*L)/(3*B2*L);
        BMD(x)=M(x)]
        ifFalse[(x >= le)
                ifTrue[M(x)=Vr*(L-x);BMD(x)=M(x)]]]

action(BMD drawing)
input(L,BMD:number)
output()
expressions

iter i=1,21
[plot BMD(i)]
```




```
action(Mmax calculation)
input(xmax: number)
output(Mmax:number)
expressions

(xmax<ls)
ifTrue[Mmax=Vl*xmax]
ifFalse[(ls =< xmax <le)
        ifTrue[Mmax=Vl*xmax-ws*(xmax-B1*L)*(xmax+B1*L)/2
           -ws*(A-1)*((xmax-B1*L)**2)*(xmax+2*B1*L)/(3*B2*L)]
        ifFalse[(x max>= le)
                ifTrue[Mmax=Vr*(L-xmax)]]


action(derivation of  shear equation)
input(L,ws,we,Vl,Vr :number)
output(SFD(x):number)
expressions

Δx = L/20
iter i = 1,21
[x = Δx*(i-1)
(x<ls)
ifTrue[V(x)=Vl; SFD(x)=V(x)]
ifFalse[(ls =< x <le)
        ifTrue[V(x) =
Vl-ws*(x-B1*L)*(1+A*((x-B1*L)**2)/(B2*L));
        SFD(x)=V(x)]
        ifFalse[(x >= le)

ifTrue[V(x)=Vr;SFD(x)=V(x)]]]

action(SFD drawing)
input( L,Vl,Vr,SFD:umber)
output()
expressions

iter i=1,21
[plot SFD(i)]
```



c. Actions.

Figure 6.35. O-type and A-type classes with actions for wide flange beam elaboration tasks (continued).

action(Vmax calculation)
input(Vl,Vr: number)
output(Vmax :number)
expressions

Vmax = max(Vl, Vr)

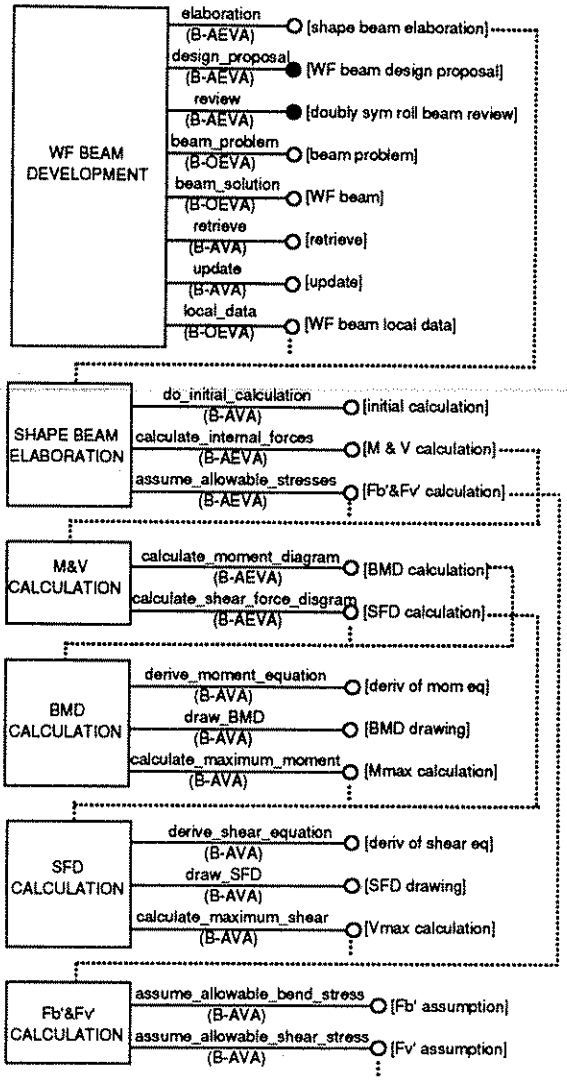
action(Fb' assumption)
input(Fy:number)
output(Fb':number)
expressions

Fb'= 0.6* Fy

action(Fv' assumption)
input(Fy:number)
output(Fv':number)
expressions

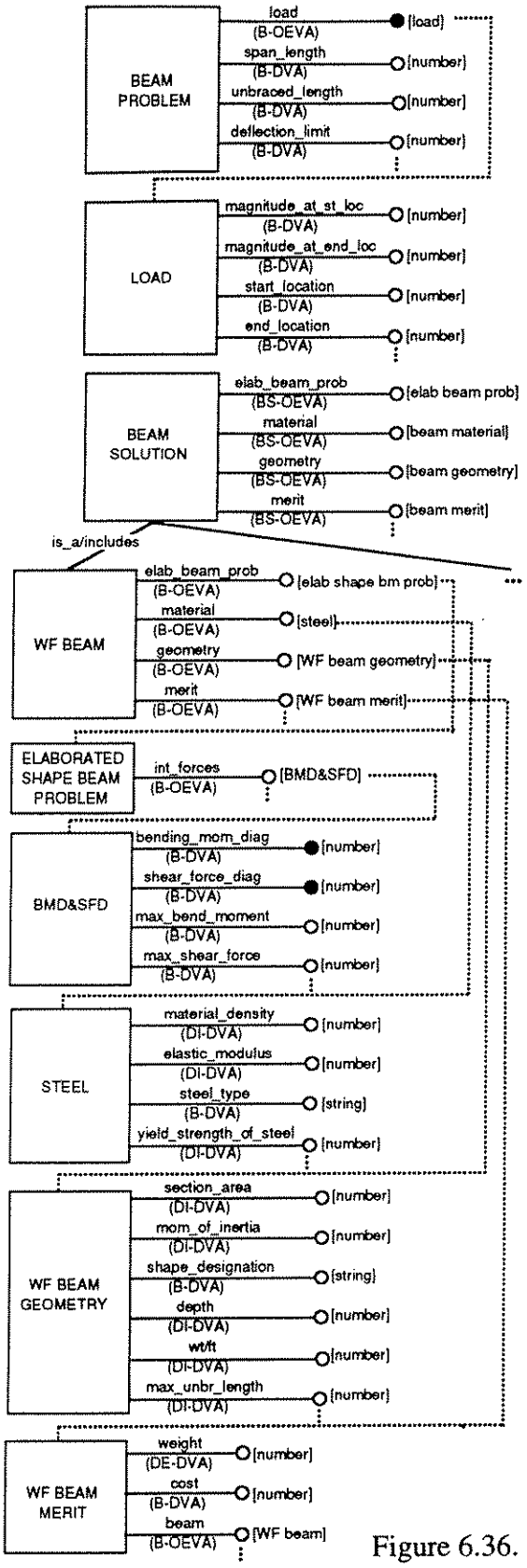
Fv'= 0.4* Fy

c. Actions (continued).

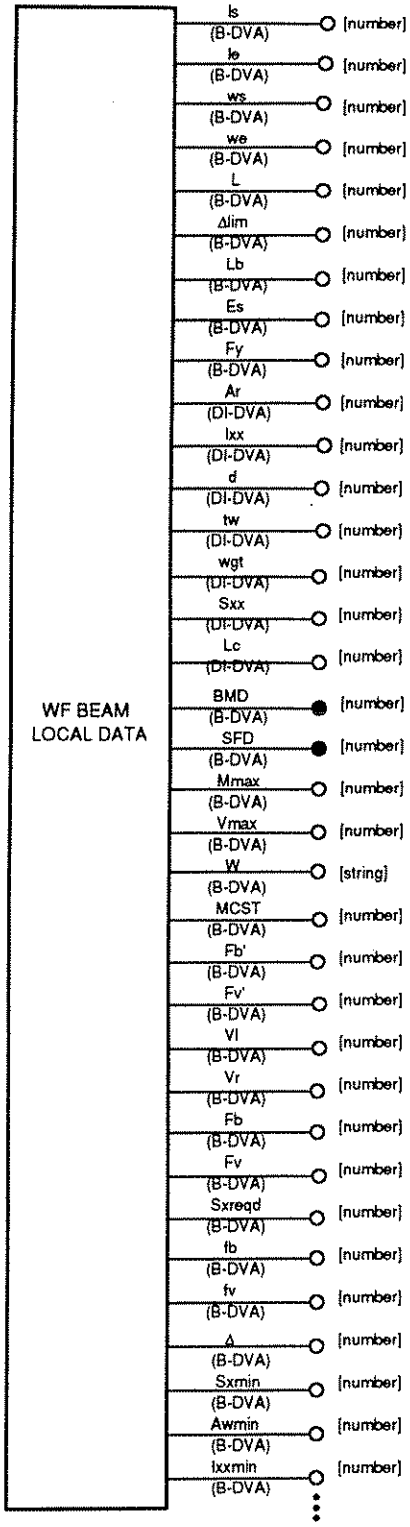


d. A-type classes.

Figure 6.35. O-type and A-type classes with actions for wide flange beam elaboration tasks (continued).



a. O-type classes.



b. WF BEAM LOCAL DATA class.

Figure 6.36. O-type and A-type classes for wide flange beam design proposal tasks.

action(Sxreqd calculation)
input(Fb',Mmax:number)
output(Sxreqd:number)
expressions

Sxreqd = Mmax/Fb'

action(shape designation selection)
input(Sxreqd :number; ASD table:array;
Sxmin,Awmin,Ixxmin,Vmax,Fv':number)
output(W:string)
expressions

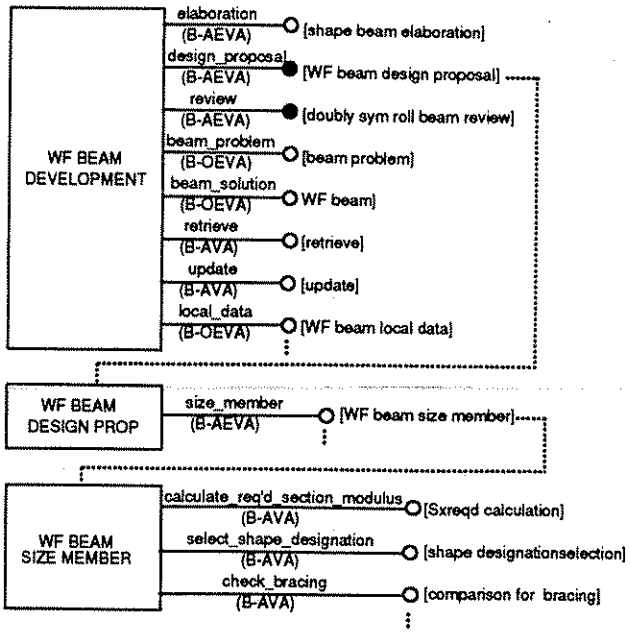
(Sxmin) isNull ifTrue[Sxmin=Sxreqd]
(Awmin) isNull ifTrue[Awmin=Vmax/Fv']
(Ixxmin) isNUll ifTrue[Ixxmin=0]

Select for W from the Allowable Stress Design Table, W must satisfy the conditions:
(Sxmin<Sxx)&
(Awmin<Aw)&
(Ixxmin<Ixx)

action(comparison for bracing)
input(Lb,Lc:number)
output(chk_rslt:string)
expressions

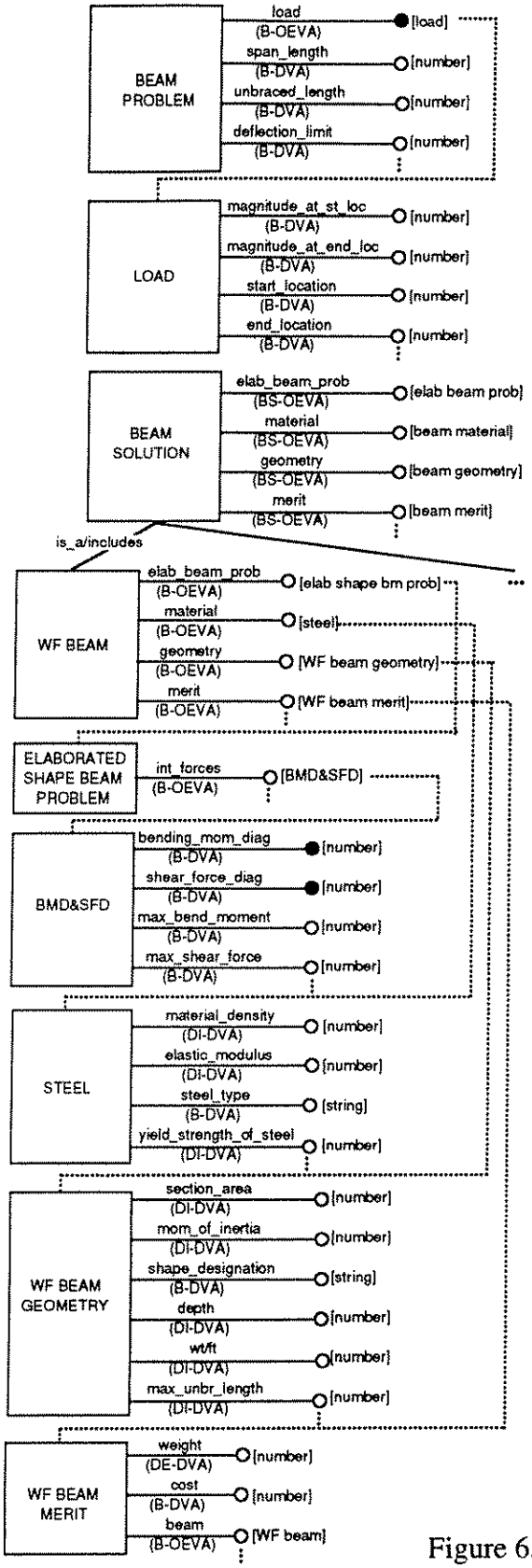
(Lb<Lc)
ifTrue[chk_rslt='OK']
ifFalse[chk_rslt='NG']

c. Actions.

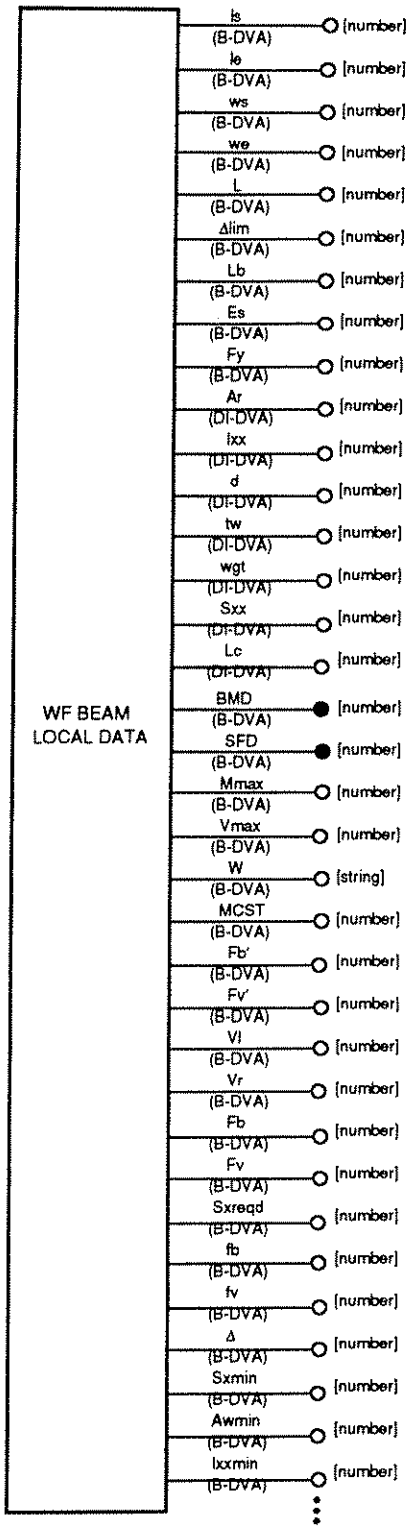


d. A-type classes.

Figure 6.36. O-type and A-type classes with actions for wide flange beam design proposal tasks (continued).



a. O-type classes.



b. WF BEAM LOCAL DATA class.

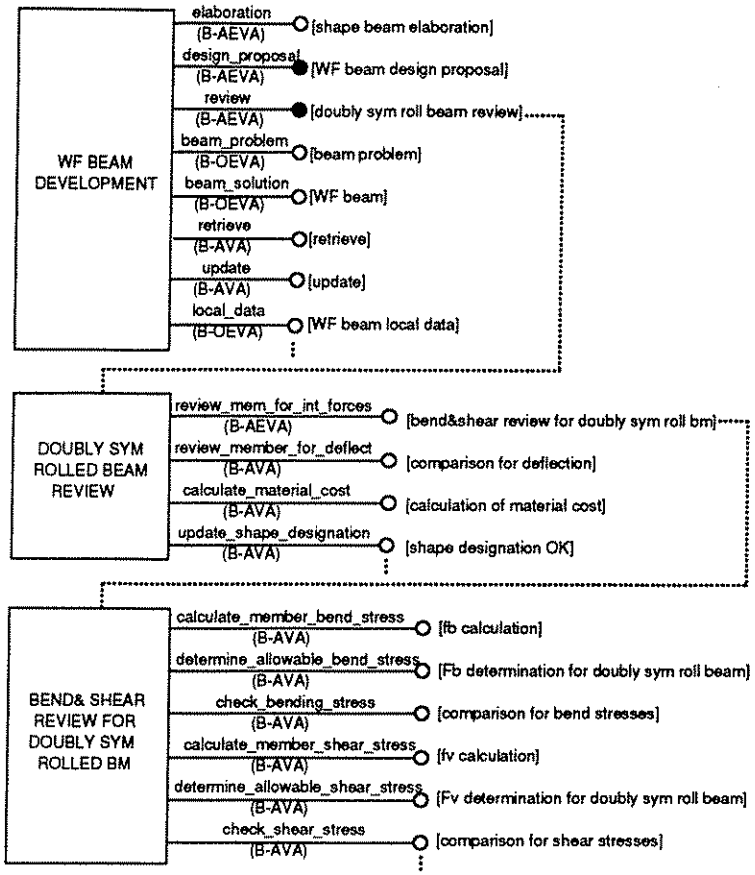
Figure 6.37. O-type and A-type classes for wide flange beam review tasks.

```
action( fb calculation)
input( Mmax,  Sxx:number)
output(fb:number)
expressions

    fb= Mmax/Sxx


action ( Fb determination for I beam)
input(Fy,bf,d,Af,Lc:number)
output(Fb:number)
expressions

 Fb from F1: number


action(fv calculation)
input(Vmax, Aw:number)
output(fv:number)
expressions

      fv= Vmax/ Aw


action (Fv determ for doubly sym rolled beam)
input( Fy, tw, d:number)
output( Fv:number)
expressions

 Fv from F4: number


action(comparison for bend stresses)
input( fb,Fb,Mmax:number)
output(chk_rslt:string; Sxmin:number)
expressions

( fb<Fb)
 ifTrue[chk_rslt='OK']
 ifFalse[chk_rslt='NG';
         Sxmin= Mmax*Fb/fb]


action(comparison for shear stresses)
input( fv,Fv,Vmax:number)
output(chk_rslt:string;Awmin:number)
expressions

( fv<Fv)
 ifTrue[chk_rslt='OK']
 ifFalse[chk_rslt='NG';
         Awmin=Vmax*Fv/fb]
```

```
action(comparison for deflection)
input(Δ,Δlim:number)
output(chk_rslt:string;Ixxmin:number)
expressions

( Δ<Δlim)
 ifTrue[chk_rslt='OK']
 ifFalse[chk_rslt='NG';
         Ixxmin=Ixx*Δ/Δmin]


action(calculation of material cost)
input(WGT, unit_price:number)
output(MCST:number)
expressions

MCST=WGT*unit_price


 action(shape designation OK)
 input(fb,Fb,fv,Fv,Δ,Δlim:number)
 output(chk_rslt:string)
 expressions

 (fb<Fb)&(fv<Fv)&(Δ<Δlim)
 ifTrue[chk_rslt='OK']
 ifFalse[chk_rslt='NG']
```

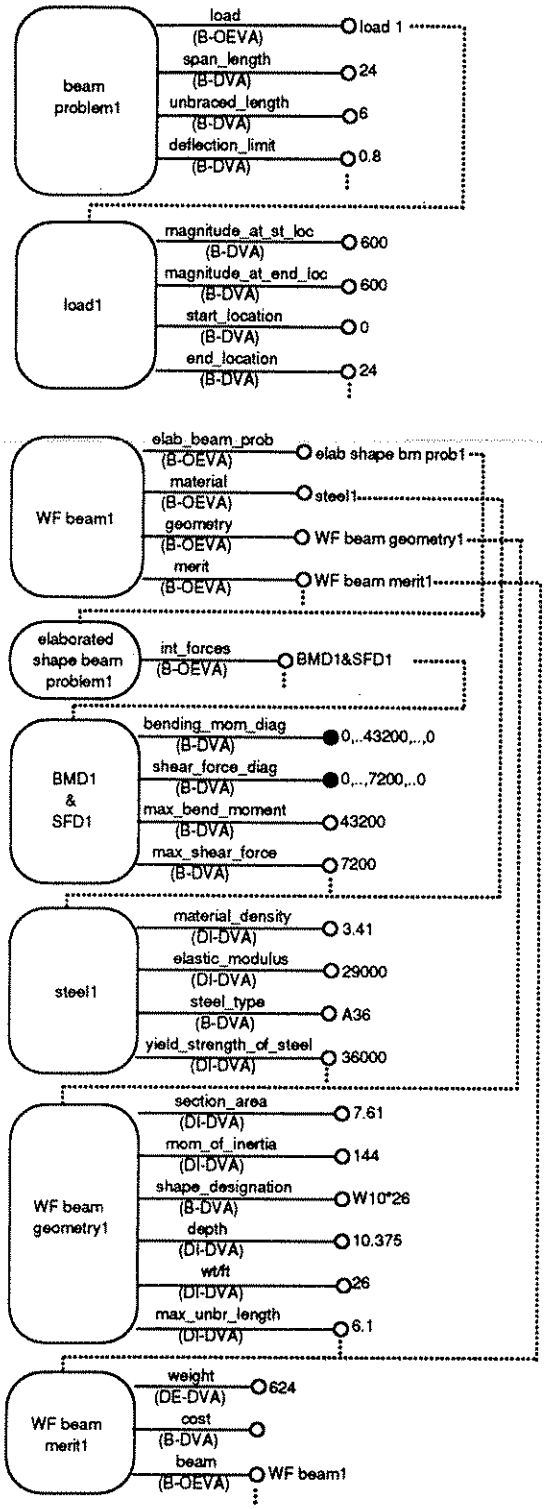
c. Actions.

Figure 6.37. O-type and A-type classes
for wide flange beam review tasks (continued).

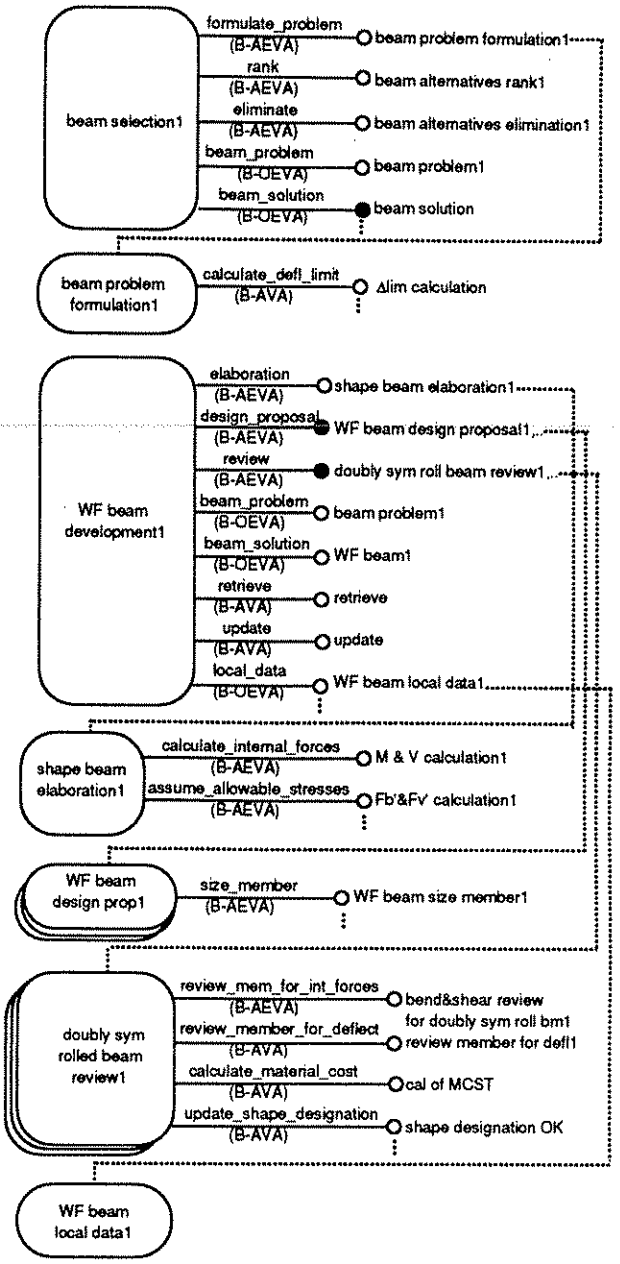


d. A-type classes.

Figure 6.37. O-type and A-type classes
for wide flange beam review tasks (continued).

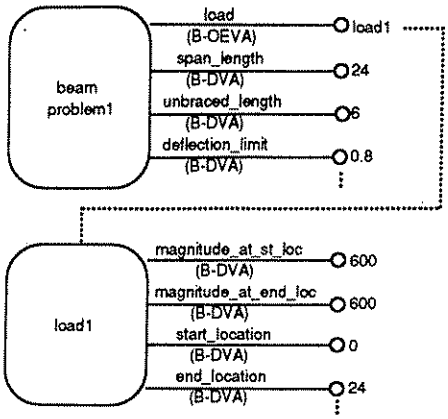


a. O-type instances for wide flange beam design.

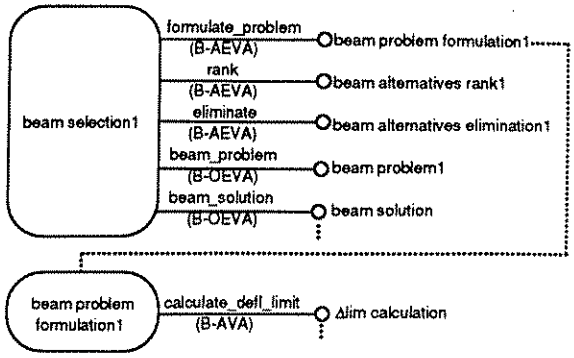


b. A-type instances for wide flange beam design.

Figure 6.38. O-type and A-type instances for wide flange beam design.



a. O-type instances.



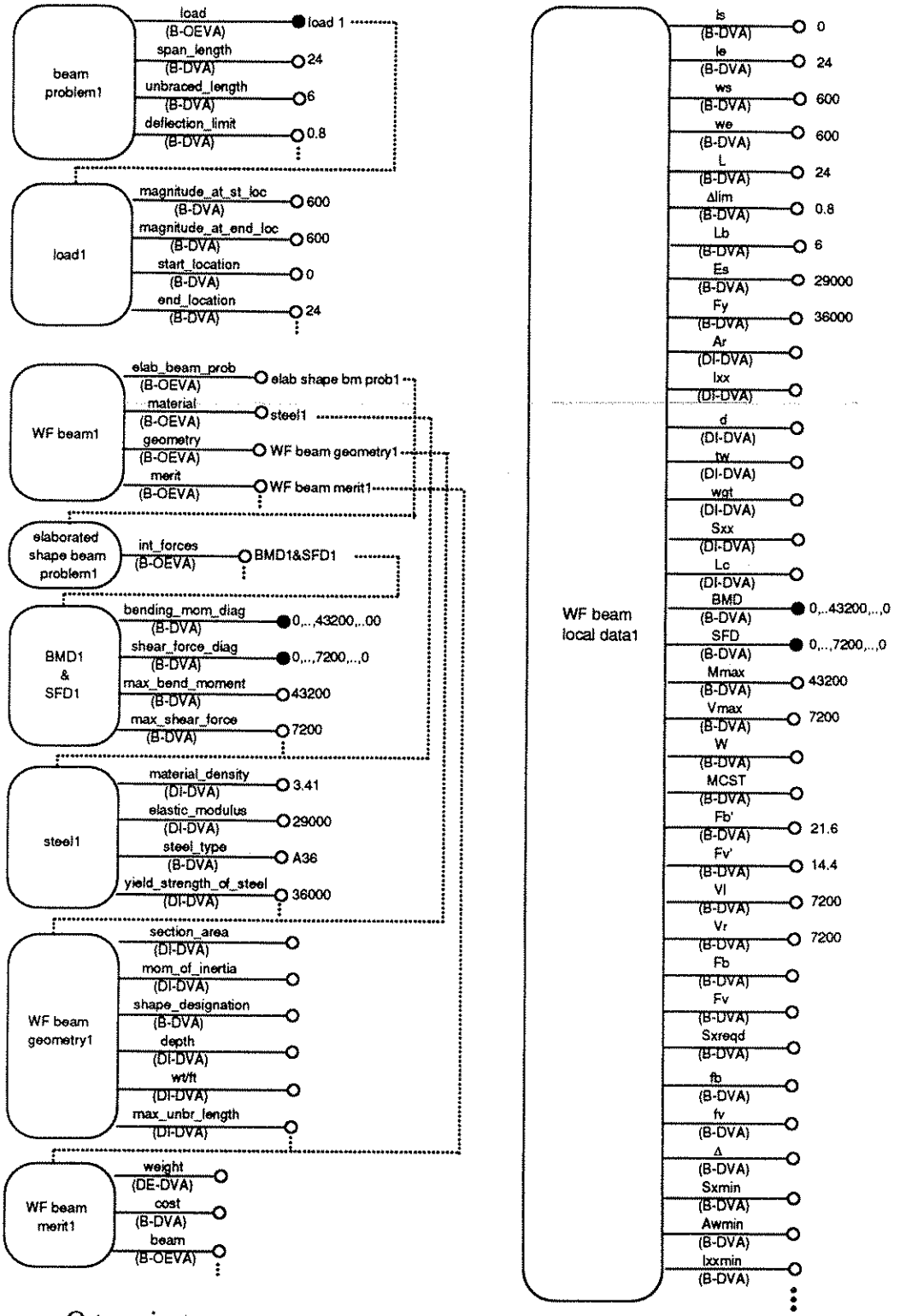
c. A-type instances.

action(Δlim calculation)
input(L:24)
output(Δlim:0.8)
expressions

L=beam_problem.span_length
Δlim=L/360
beam_problem.deflection_limit=Δlim

b. Actions.

Figure 6.39. O-type and A-type instances for wide flange beam selection tasks.

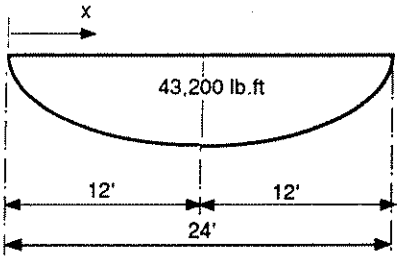


a. O-type instances.

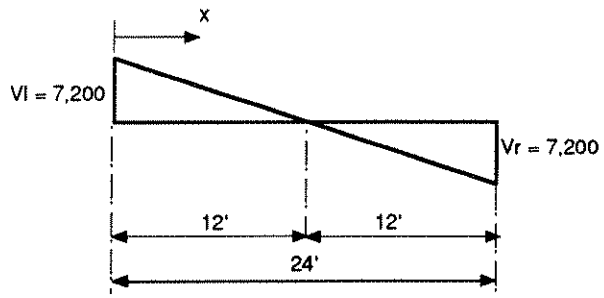
b. The WF beam local data1 instance.

Figure 6.40. O-type and A-type instances for wide flange beam elaboration tasks.

```
action(initial calculation)
input(L:24,ls:0,le:24,we:600,ws:600)
output(B1:0,B2:1,B3:0,A:1,Vl:7200 ,Vr:7200)
expressions

B1 = ls/L
B2 = (le-ls)/L
B3 = 1-le/L
A = we/ws
Vl = ws*B2*L((2+A)*B2+3*(A+1)*B3)/6
Vr = ws*B2*L(3*(A+1)*(1-B3)-(2+A)*B2)/6

action(derivation of moment equation)
input(L:24,ws:600, we:600, Vl:7200,Vr:7200,A:1)
output(BMD(x):0,...,43200,...,0)
expressions

Δx = L/20
iter i = 1,21
[x = Δx*(i-1)
(x<ls)
ifTrue[M(x)=Vl*x; BMD(x)=M(x)]
ifFalse[(ls =< x <le)
        ifTrue[M(x) = V*x-ws*(x-B1*L)*(x+B1*L)/2
       -ws*(A-1)*((x-B1*L)**2)*(x+2*B1*L)/(3*B2*L);
        BMD(x)=M(x)]
        ifFalse[(x >= le)
                ifTrue[M(x)=Vr*(L-x);BMD(x)=M(x)]]]

action(BMD drawing)
input(L:24,BMD:0,..,43200,...,0)
output()
expressions

iter i=1,21
[plot BMD(i)]
```




```
action(Mmax calculation)
input(xmax:12)
output(Mmax:43200)
expressions

(xmax<ls)
ifTrue[Mmax=Vl*xmax]
ifFalse[(ls =< xmax <le)
        ifTrue[Mmax=Vl*xmax-ws*(xmax-B1*L)*(xmax+B1*L)/2
           -ws*(A-1)*((xmax-B1*L)**2)*(xmax+2*B1*L)/(3*B2*L)]
        ifFalse[(x max>= le)
                ifTrue[Mmax=Vr*(L-xmax)]]


action(derivation of  shear equation)
input(L:24,ws:600,we:600,Vl:7200,Vr:7200)
output(SFD(x):0,..,7200,..,0)
expressions

Δx = L/20
iter i = 1,21
[x = Δx*(i-1)
(x<ls)
ifTrue[V(x)=Vl; SFD(x)=V(x)]
ifFalse[(ls =< x <le)
        ifTrue[V(x) =
Vl-ws*(x-B1*L)*(1+A*((x-B1*L)**2)/(B2*L));
        SFD(x)=V(x)]
        ifFalse[(x >= le)
                ifTrue[V(x)=Vr;SFD(x)=V(x)]]]


action(SFD drawing)
input(L:24,Vl:7200,Vr:7200,SFD:0,...,7200,...,0)
output()
expressions

iter i=1,21
[plot SFD(i)]
```



c. Actions.

Figure 6.40. O-type and A-type instances with actions for wide flange beam elaboration tasks (continued).

action(Vmax calculation)
input(Vl:7200,Vr:7200)
output(Vmax:7200)
expressions

Vmax = max(Vl, Vr)

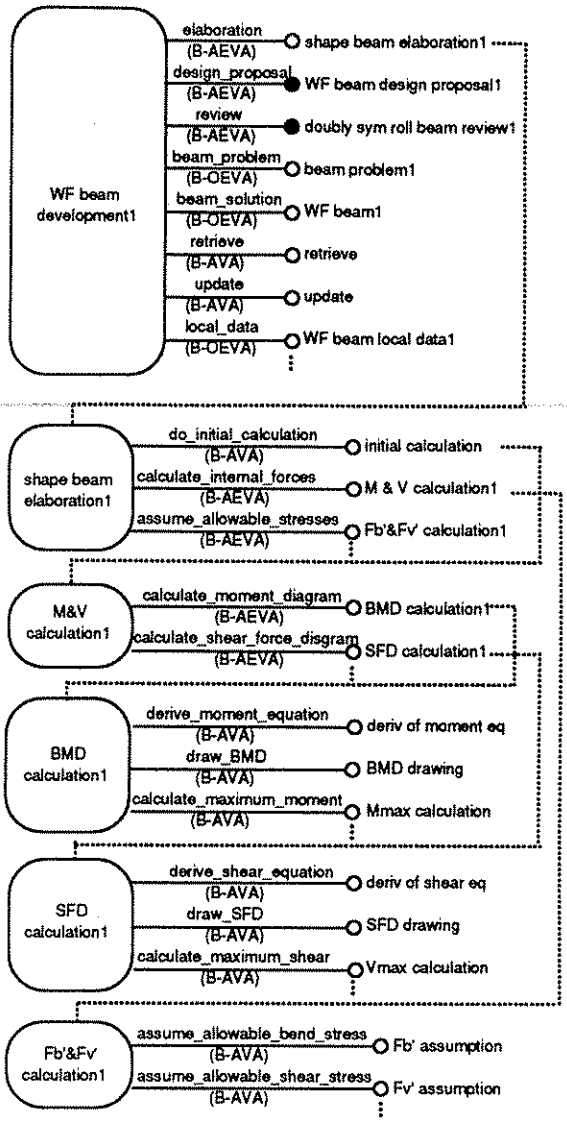
action(Fb' assumption)
input(Fy:36)
output(Fb':21.6)
expressions

Fb'= 0.6* Fy

action(Fv' assumption)
input(Fy:36)
output(Fv':14.4)
expressions

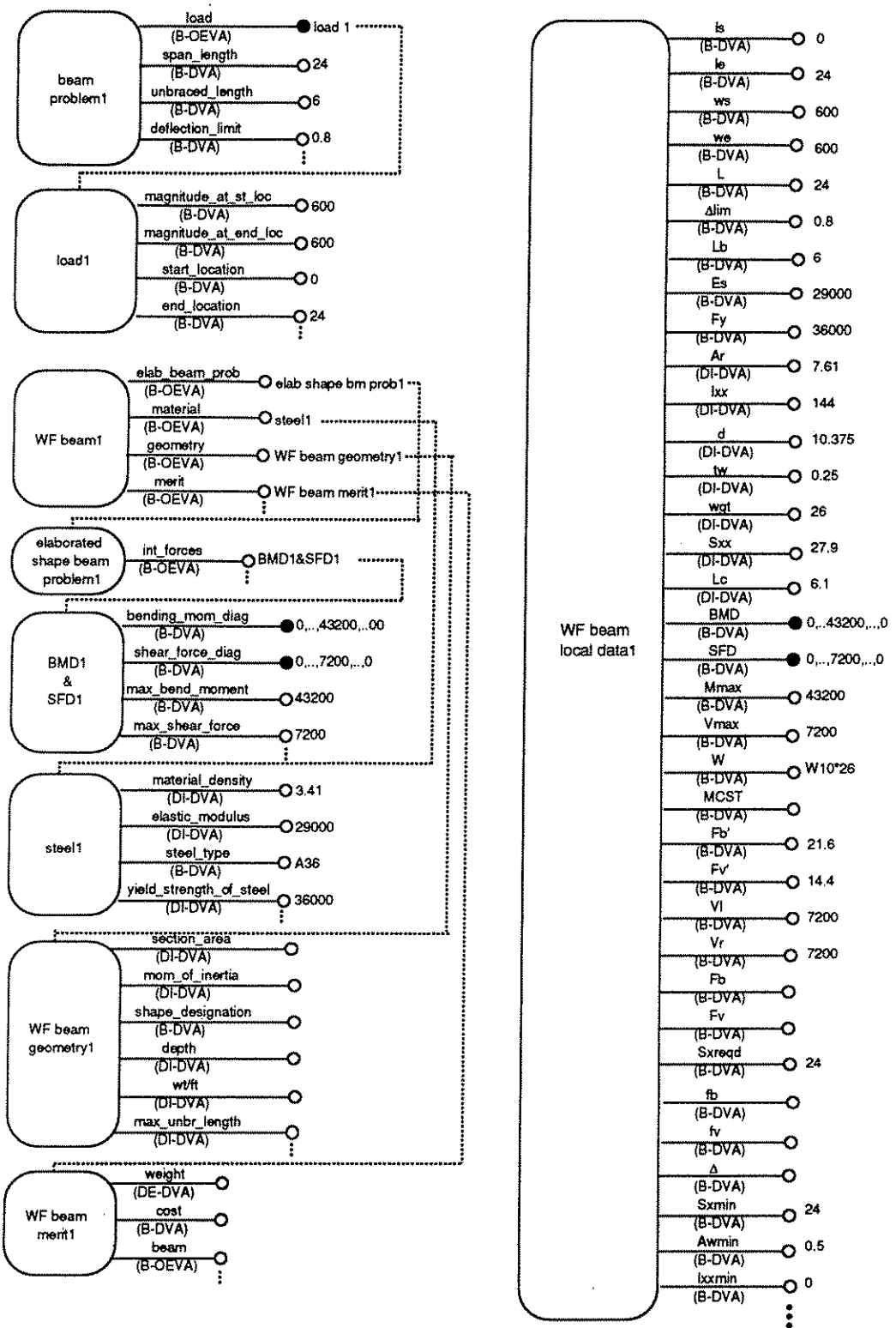
Fv'= 0.4* Fy

c. Actions (continued).



d. A-type instances.

Figure 6.40. O-type and A-type instances with actions for wide flange beam elaboration tasks (continued).



a. O-type instances. b. The WF beam local data1 instance.

Figure 6.41. O-type and A-type instances for wide flange beam design proposal tasks.

action(Sxreqd calculation)
input(Fb':21.6,Mmax:43200)
output(Sxreqd:24)
expressions

Sxreqd = Mmax/Fb'

action(shape designation selection)
input(Sxreqd:24; ASD table:array;Sxmin:0,
Awmin:0,Ixxmin:0,Vmax:7200,Fv':14.4)
output(W:W10*26)
expressions

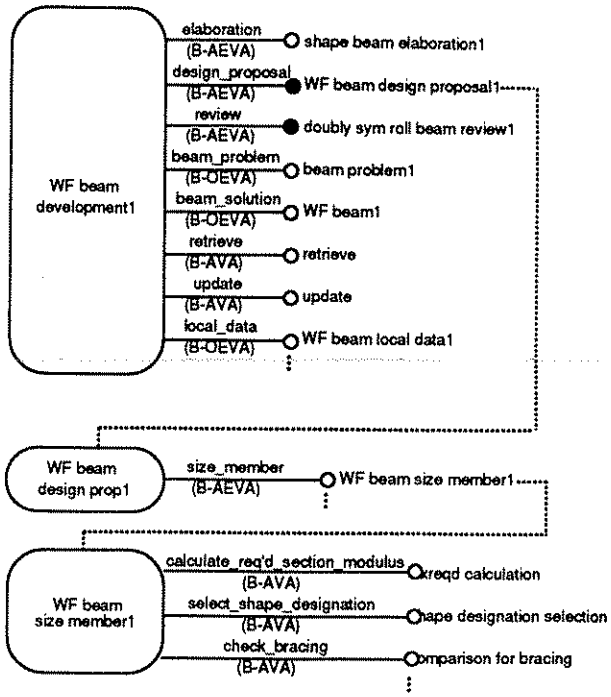
(Sxmin) isNull ifTrue[Sxmin=Sxreqd]
(Awmin) isNull ifTrue[Awmin=Vmax/Fv']
(Ixxmin) isNUll ifTrue[Ixxmin=0]

Select for W from the Allowable Stress Design Table,
W must satisfy the conditions:
(Sxmin<Sxx)&
(Awmin<Aw)&
(Ixxmin<Ixx)

action(comparison for bracing)
input(Lb:6,Lc:6.1)
output(chk_rslt:OK)
expressions

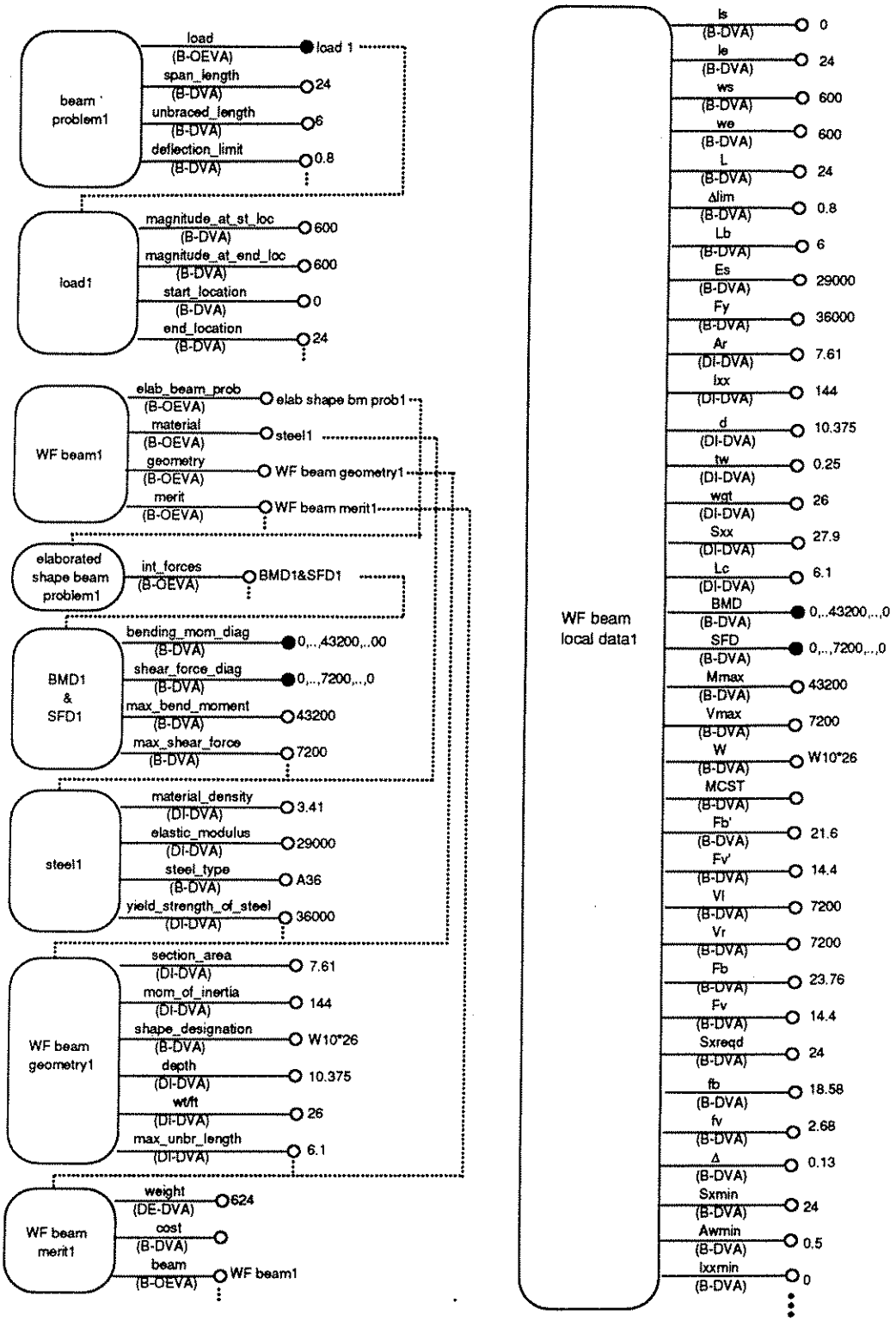
(Lb<Lc)
ifTrue[chk_rslt='OK']
ifFalse[chk_rslt='NG']

c. Actions.



d. A-type instances.

Figure 6.41. O-type and A-type instances
for wide flange beam design proposal tasks (continued).



a. O-type instances. b. The WF beam local data1 instance.

Figure 6.42. O-type and A-type instances for wide flange beam review tasks.

action(fb calculation)
input(Mmax:43200, Sxx:27.9)
output(fb:18.58)
expressions

fb= Mmax/Sxx

action (Fb determination for I beam)
input(Fy:36,bf:5.77,d:10.33,Af:4.54,Lc:6.1)
output(Fb:23.76
expressions

Fb from F1: number

action(fv calculation)
input(Vmax:7200, Aw:2.67)
output(fv:2.68)
expressions

fv= Vmax/ Aw

action (Fv determ for doubly sym rolled beam)
input(Fy:36, tw0.25, d:10.375)
output(Fv:14.4)
expressions

Fv from F4: number

action(comparison for bend stresses)
input(fb:18.58,Fb:23.76,Mmax:43200)
output(chk_rslt:OK;Sxmin:0)
expressions

(fb<Fb)
ifTrue[chk_rslt='OK']
ifFalse[chk_rslt='NG';
Sxmin= Mmax*Fb/fb]

action(comparison for shear stresses)
input(fv:2.68,Fv:14.4,Vmax:7200)
output(chk_rslt:OK;Awmin:0)
expressions

(fv<Fv)
ifTrue[chk_rslt='OK']
ifFalse[chk_rslt='NG';
Awmin=Vmax*Fv/fb]

action(comparison for deflection)
input(Δ:0.13,Δlim:0.8)
output(chk_rslt:OK;Ixxmin:0)
expressions

(Δ<Δlim)
ifTrue[chk_rslt='OK']
ifFalse[chk_rslt='NG';
Ixxmin=Ixx*Δ/Δmin]

action(calculation of material cost)
input(WGT, unit_price:number)
output(MCST:number)
expressions

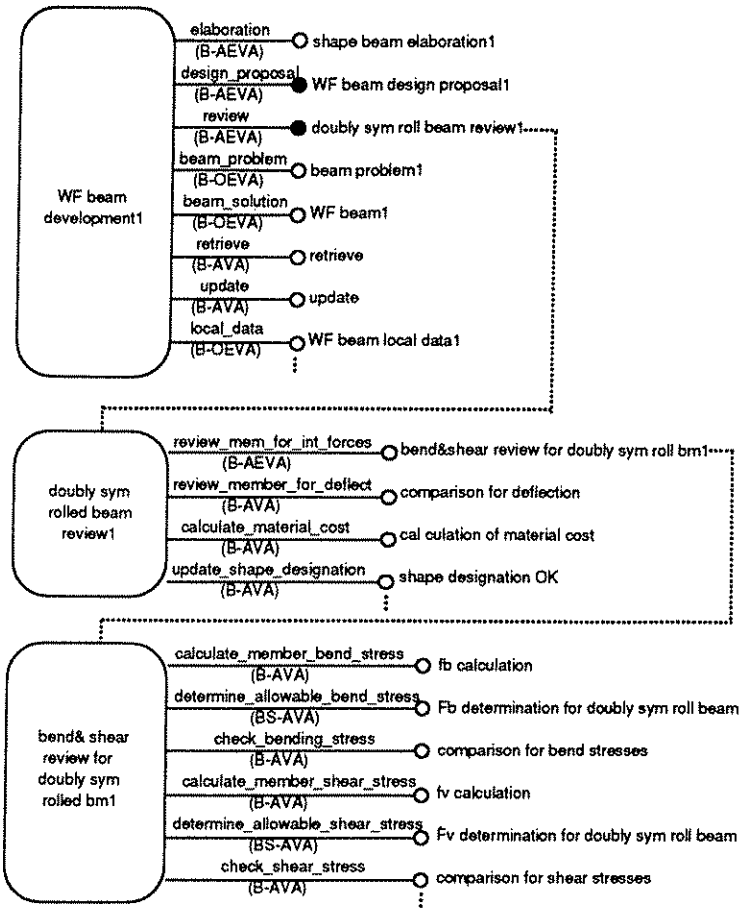
MCST=WGT*unit_price

action(shape designation OK)
input(fb:18.58,Fb:23.76,fv:2.68,Fv:14.4,Δ:0.13,Δlim:0.8)
output(chk_rslt:OK)
expressions

(fb<Fb)&(fv<Fv)&(Δ<Δlim)
ifTrue[chk_rslt='OK']
ifFalse[chk_rslt='NG']

c. Actions.

Figure 6.42. O-type and A-type instances
for wide flange beam review tasks (continued).



d. A-type instances.

Figure 6.42. O-type and A-type instances for wide flange beam review tasks (continued).

References

Abdalla, J.A.; Powell, G.H. (1991), "Version Management Needs for Structural Engineering Design," Engineering with Computers, 7, pp. 131-143.

AISC (1989), Manual of Steel Construction: Allowable Stress Design, Ninth Edition, American Institute of Steel Construction, Inc., Chicago.

Allen, R.H. (1992), Expert Systems for Civil Engineering: Knowledge Representation, American Society of Civil Engineers, N.Y., New York.

Asimow, M. (1962), Introduction to Design, Prentice-Hall, Inc., Englewood Cliffs, NJ.

Batini, C.; Ceri, S.; Navathe, S. (1992), Conceptual Database Design: An Entity-Relationship Approach, Benjamin/Cummings, Redwood City, CA, First Edition.

Bjork, B-C. (1989), "Basic Structure of a Proposed Building Product Model," CAD, 21 (2), 71-78.

Booch, G. (1986), Object-Oriented Design, Benjamin/Cummings, Redwood City, CA.

Chen, P.P.S. (1976), "The Entity-Relationship Model-Toward a Unified View of Data," ACM Transactions on Database Systems 1, 1, pp. 9-36.

Date, C. J. (1981), An Introduction to Database Systems, Addison-Wesley Publishing Company, Third Edition.

Dym, C.L.; Levitt, R.E. (1991), "Toward the Integration of Knowledge for Engineering Modeling and Computation," Engineering with Computers, 7, pp. 209-224.

Eastman, C.M.; Bond, A.H.; Chase, S.C. (1991), "A Formal Approach for Product Model Information," Research in Engineering Design, 2, pp. 65-80.

Eastman, C.M.; Bond, A.H.; Chase, S.C. (1991), "Application and Evaluation of an Engineering Data Model," Research in Engineering Design, 2, pp. 185-207.

Efraim, T. (1992) Expert Systems and Applied Artificial Intelligence, Macmillan Publishing Company, New York.

Fenves, G.L. (1990), "Object-Oriented Programming for Engineering Software Development," Engineering with Computers, 6, pp. 1-15.

Garrett Jr., James H.; Hakim, M.M. (1992), "Object-Oriented Model of Engineering Design Standards," Journal of Computing in Civil Engineering, ASCE, 6 (3), pp. 323-347.

Gielingh (1988), "General AEC reference model (GARM)," Technical Report, AEC committee of ISO/STEP, Delft, Netherlands.

Hammer, M.; McLeod, D. (1981), "Database Description with SDM: A Semantic Database Model," ACM Transactions on Database Systems, Vol. 6, No. 3, pp. 351-386.

Howard, H.C. (1991), "Project-Specific Knowledge Bases in AEC Industry," Journal of Computing in Civil Engineering, 5 (1), ASCE, pp. 25-41.

Howard, H.C.; Abdalla, J.A., Phan, D.H.; Lavakane, A.P. (1991), "Primitive-Composite Approach for Structural Data," Computing in Civil Engineering and Symposium on Data Bases, Proceedings of the Seventh Conference, ASCE, pp. 799-808.

Hull, R.; King, R. (1987), "Semantic Database Modeling: Survey, Applications, and Research Issues," ACM Computing Surveys, 19 (3), pp. 201-260.

IPO (1991), IGES/PDES Organization Reference Manual, National Computer Graphics Association, Fairfax, VA.

Jain, D.; Krawinkler, H.; Law, K.H; Luth, G.P. (1991), "A Formal Approach to Automating Conceptual Structural Design, Part I: Methodology," Engineering with Computers, 7, pp. 91-107.

Kim, W. (1990), Introduction to Object-Oriented Databases, MIT Press.

Lalonde, W.R.; Pugh, J.R. (1990), Inside SmallTalk, Vol. 1, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Law, K.H.; Barsalou, T.; Wiederhold, G. (1990), "Management of Complex Structural Engineering Objects in a Relational Framework," Engineering with Computers, 6, pp. 81-92.

Lee, H.H.; Arora, J.S. (1991), "Object-Oriented Programming for Engineering Applications," Engineering with Computers, 7, pp. 225-235.

Lin, T.Y.; Stotesbury, S.D. (1988), Structural Concepts and Systems for Architects and Engineers, Van Nostrand Reinhold Co., New York.

Luth, G.P.; Jain, D.; Krawinkler, H.; Law, K.H. (1991), "A Methodology for Automating Conceptual Structural Design," Computing in Civil Engineering and Symposium on Data Bases, Proceedings of the Seventh Conference, ASCE, pp. 34-43.

Luth, G.P.; Jain, D.; Krawinkler, H.; Law, K.H. (1991), "A Formal Approach to Automating Conceptual Structural Design, Part I: Methodology," Engineering with Computers, 7, pp. 79-89.

Madden, J.A.; Sause, R. (1992), "A Design Product Model for Computer Integrated Structural Engineering," Computing in Civil Engineering and Geographic Information Systems Symposium, Proceedings of the Eighth Conference , ASCE, pp. 113-120.

Madden, J.A. (1992), " Product and Process Models for Computer Integrated Preliminary Structural Design," Master's Thesis, Lehigh University.

Martini, K.; Sause, R.; Powell, G.H. (1991), " Models for Computer Integrated Civil Engineering Design," The Fourth International Conference on Computing in Civil and Building Engineering: Extended Abstracts, Tokyo, Japan.

Nevill Jr., G.E.; Garcelon, J.H.; Tambling, K.B. (1989), "Creating Abstraction/Feature Sets for Top Down Mechanical Design: Experiences with Automating Preliminary Design of Structures,"

Computers in Engineering, ASME, pp. 273-279.

Phan, D.H.D.; Abdalla, J.A.; Howard, H.C. (1992), "An Integrated Representation of Form, Function and Behavior in Structural Engineering," Computing in Civil Engineering and Geographic Information Systems Symposium, Proceedings of the Eighth Conference, ASCE, pp. 394-401.

Phan, D.H.D.; Howard, H.C. (1994), "Functional Analysis using Partitioned Engineering Data Flow Model," Journal of Computing in Civil Engineering, ASCE, 8 (1), pp. 2-19.

Powell, G.H.; Bhateja, R. (1988), "Data Base Design for Computer-Integrated Structural Engineering," Engineering with Computers, 4, pp. 135-143.

Ross, T.J.; Morrow, J.P.; Wagner, L.R.; Luger, G.F. (1992), "Two Paradigms for OOP Models for Scientific Applications," Computing in Civil Engineering and Geographic Information Systems Symposium, Proceedings of the Eighth Conference, ASCE, pp. 535-542.

Rumbaugh, J.; Blaha, M.; Premerlani, W.; Eddy, F.; Lorensen, W. (1991), Object-Oriented Modeling and Design, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Salustri, F.A.; Venter, R.D. (1992), "An Axiomatic Theory of Engineering Design Information," Engineering with Computers, 8, pp. 197-211.

Sanvido, V.E.; Kumara, S.; Ham, I. (1989), "A Top-Down Approach to Integrating the Building Process," Engineering with Computers, 5, pp. 91-103.

Sanvido, V.E.; Fenves, S.J.; Wilson, J.L. (1992), "Towards a Virtual Master Builder," Journal of Professional Issues, ASCE, 118 (3), pp. 261-278.

Sanvido, V.E.; Messner, J. (1992), "Classifying Process Control Information," Computing in Civil Engineering and Geographic Information Systems Symposium, Proceedings of the Eighth Conference, ASCE, pp. 340-347.

Sause, R. (1989), "A Model of the Design Process for Computer Integrated Structural

Engineering", Ph.D. Dissertation, University of California, Berkeley.

Sause, R.; Powell, G.H. (1990), "A Design Process Model for Computer Integrated Structural Engineering," Engineering with Computers, 6, 3, pp. 129-143.

Sause, R.; Powell, G.H. (1991), "A Design Process Model for Computer Integrated Structural Engineering: Design Phases and Tasks," Engineering with Computers, 7, pp. 145-160.

Sause, R. (1991), "Towards Management of Design Alternatives in Object Oriented Databases," Computing in Civil Engineering and Symposium on Data Bases, Proceedings of the Seventh Conference, ASCE, pp. 202-211.

Sause, R.; Martini, K.; Powell, G.H. (1992), "Object Oriented Approaches for Integrated Engineering Design Systems," Journal of Computing in Civil Engineering, ASCE, 6 (3), pp. 248-265.

Sause, R. and Madden J.A. (1993), "Toward Integrated Structural Engineering Design Models," Information Technology for Civil and Structural Engineers, Proceedings of Civil-Comp 93, the Fifth International Conference on Civil and Structural Engineering Computing, Civil-Comp Press, pp. 115-120.

Smith, J.M.; Smith, D.C.P. (1977), "Database Abstractions: Aggregation," Communications of the ACM , 20 (6), pp. 405-413.

Smith, J.M.; Smith, D.C.P. (1977), "Database Abstractions: Aggregation and Generalization," ACM Transactions on Database Systems, 2 (2), pp. 105-133.

Thompson, J. P. (1989), Data with Semantics: Data Models and Data Management, Van Nostrand Reinhold, New York, New York.